FREE DUAL-CORE MICROCONTROLLER

# HackSpace

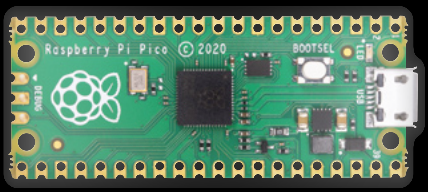
TECHNOLOGY IN YOUR HANDS

hsmag.cc | February 2021 | Issue #39

## INTRODUCING

RASPBERRY PI

# PICO


Raspberry Pi Pico © 2020
BOOTSEL

LOW-COST · HIGH-PERFORMANCE · FLEXIBLE I/O

**A NEW MICROCONTROLLER BOARD FROM RASPBERRY PI**

9 772515 514006 39
Raspberry Pi PRESS
Feb. 2021
Issue #39 £6

VISIT…

LANZAROTE
Caliente.com

OKDO™
DESIGN THE WORLD

CHECK OUT THE NEW OKDO PROJECT HUB!

Find out more at www.okdo.com/projecthub

MAKE A SIMPLE GAME CONTROLLER
214
Difficulty: Easy

MOTION ACTIVATED CAMERA
442
Difficulty: Moderate

STREAMING SERVER
82
Difficulty: Difficult

Find inspiring projects created by our customers and engineers that show you how to build and programme cool projects and how to get started with some of your favourite boards!

Find out more about our full product range at:

www.okdo.com

@letsokdo

# Welcome to HackSpace magazine

We've been waiting for months to be able to shout about this issue, so it's exciting to finally be able to talk about it. Pico is the latest member of the Raspberry Pi stable and, if you've got a paper version of this magazine, you should now be holding one in your hands. It's a thoroughly modern board, designed to bring the Raspberry Pi ethos to the world of microcontrollers. It's cheap ($4), powerful (dual core running at up to 133MHz), and comes with an innovative Programmable Input/Output system that lets it do amazingly fast I/O.

We've got a whole load of information on how to get started with this, starting on page 30, but for a sneak peek at what you can achieve, take a look at page 14 for a game (inspired by the 1980s classic Zarch) programmed by Eben Upton, and ported to Pico by Graham Sanderson.

We look forward to seeing what you create with your Picos. Send pictures of your creations to **hackspace@raspberrypi.com** or tag **@HackSpaceMag** on Twitter.

**BEN EVERARD**
**Editor** ben.everard@raspberrypi.com

PAGE 46
FREE PICO
WHEN YOU SUBSCRIBE

FREE DUAL-CORE MICROCONTROLLER
HackSpace
TECHNOLOGY IN YOUR HANDS
INTRODUCING
RASPBERRY PI
PICO
LOW-COST · HIGH-PERFORMANCE · FLEXIBLE I/O
A NEW MICROCONTROLLER BOARD FROM RASPBERRY PI

Got a comment, question, or thought about HackSpace magazine?

get in touch at
**hsmag.cc/hello**

## GET IN TOUCH

hackspace@raspberrypi.com
hackspacemag
hackspacemag

## ONLINE

hsmag.cc

Available on the App Store

GET IT ON Google Play

## EDITORIAL

**Editor**
Ben Everard
ben.everard@raspberrypi.com

**Features Editor**
Andrew Gregory
andrew.gregory@raspberrypi.com

**Sub-Editors**
David Higgs, Nicola King

## DESIGN

**Critical Media**
criticalmedia.co.uk

**Head of Design**
Lee Allen

**Designers**
Sam Ribbits, James Legg, Ty Logan

**Photography**
Brian O'Halloran

## CONTRIBUTORS

Lucy Rogers, Drew Fustini, Jo Hinchliffe, Mayank Sharma, Gareth Halfacree, Emily Velasco, Marc de Vinck, PJ Evans, Helen Leigh

## PUBLISHING

**Publishing Director**
Russell Barnes
russell@raspberrypi.com

**Advertising**
Charlie Milligan
charlotte.milligan@raspberrypi.com

## DISTRIBUTION

Seymour Distribution Ltd
2 East Poultry Ave,
London EC1A 9PT
+44 (0)207 429 4000

## SUBSCRIPTIONS

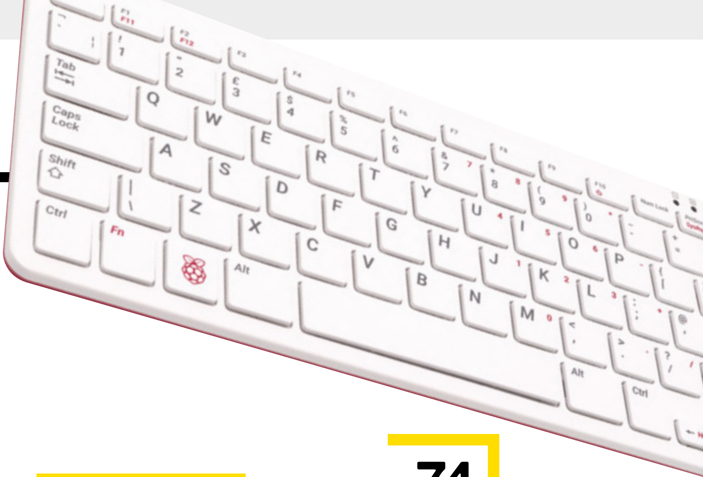
Unit 6, The Enterprise Centre,
Kelvin Lane, Manor Royal,
Crawley, West Sussex, RH10 9PE

**To subscribe**
01293 312189
hsmag.cc/subscribe

**Subscription queries**
hackspace@subscriptionhelpline.co.uk

This magazine is printed on paper sourced from sustainable forests. The printer operates an environmental management system which has been assessed as conforming to ISO 14001.

HackSpace magazine is published by Raspberry Pi (Trading) Ltd., Maurice Wilkes Building, St. John's Innovation Park, Cowley Road, Cambridge, CB4 0DS The publisher, editor, and contributors accept no responsibility in respect of any omissions or errors relating to goods, products or services referred to or advertised. Except where otherwise noted, content in this magazine is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0). ISSN: 2515-5148.



# Contents

## 06 SPARK

**06** **Top Projects**
Special things made by human beings

**20** **Objet 3d'art**
Brilliant engineering to fetch power from plastic

**22** **Columns**
Reuse, recycle, remake, remodel

**24** **Letters**
Send us your musings on making

**26** **Kickstarting**
A hydroponics platform powered by MicroPython

## 29 LENS

**30** **Raspberry Pi Pico**
Say hello to the new microcontroller from Raspberry Pi

**48** **How I Made:** Optical sound decoder
Squeezing sound out of squiggles on tape

**54** **Interview:** Eben Upton
The boss of Raspberry Pi on why Pico exists

**62** **Improviser's Toolbox** Magnets
Add clean stickiness to anything

74

### Cover Feature

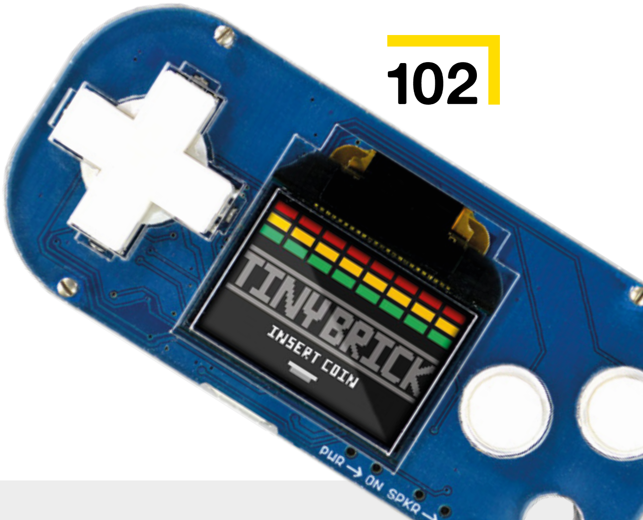
RASPBERRY PI
PICO
LOW-COST • HIGH-PERFORMANCE • FLEXIBLE I/O
Raspberry Pi Pico © 2020
BOOTSEL
LED
USB
DEBUG

30

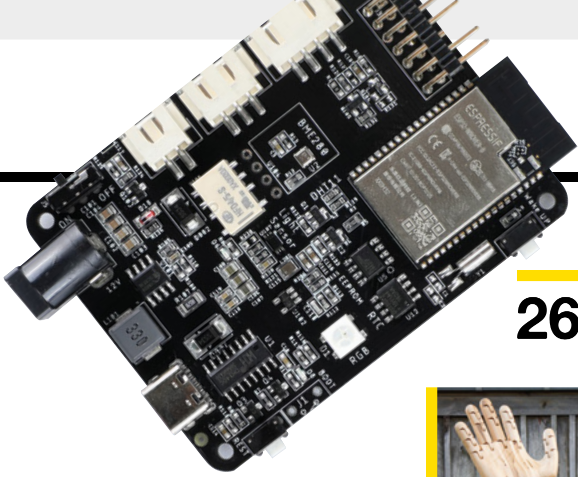
### Tutorial

**DIY music**

84 Build an instrument that sounds and looks unique to you


TINYBRICK
INSERT COIN
PWR → ON SPKR

102



**26**

## Direct from Shenzhen

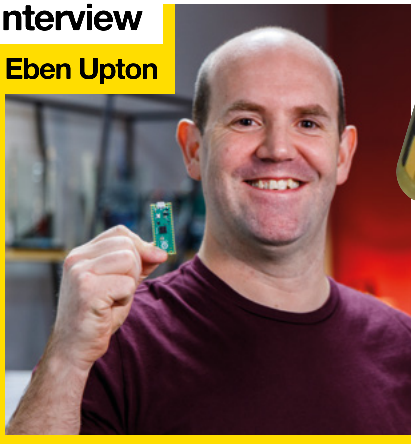
**Hot air gun**

**108** Sometimes cheap isn't cheerful – sometimes it's DANGEROUS!

**06**

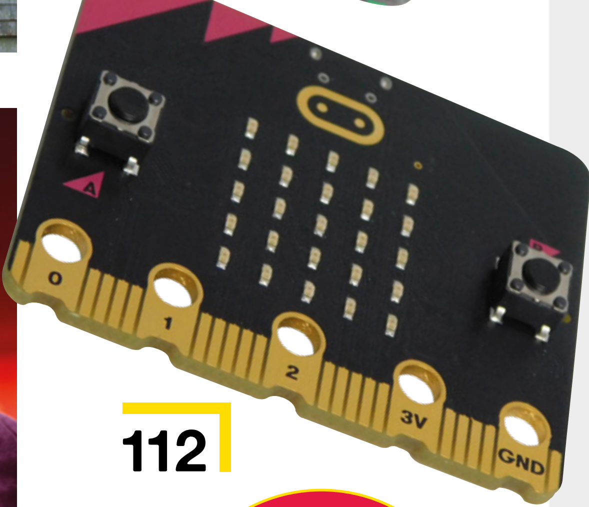
## Interview

**Eben Upton**



**54** On why the Raspberry Pi Pico came into existence

**110**



**112**

**FREE PICO** with your subscription PG 46

## 67 FORGE

68 **SoM** Traffic lights
Flash coloured LEDs on a Pico the easy way

74 **SoM** Raspberry Pi 400
More ways to play with this does-it-all machine

78 **Tutorial** FreeCAD
Design parametric parts

84 **Tutorial** DIY music
Build a capacitive touch musical sculpture

90 **Tutorial** NeoPixels with PIO
Flash coloured LEDs on a Pico the hard way

96 **Tutorial** Lasers
Add touchscreen control to a laser cutter

## 101 FIELD TEST

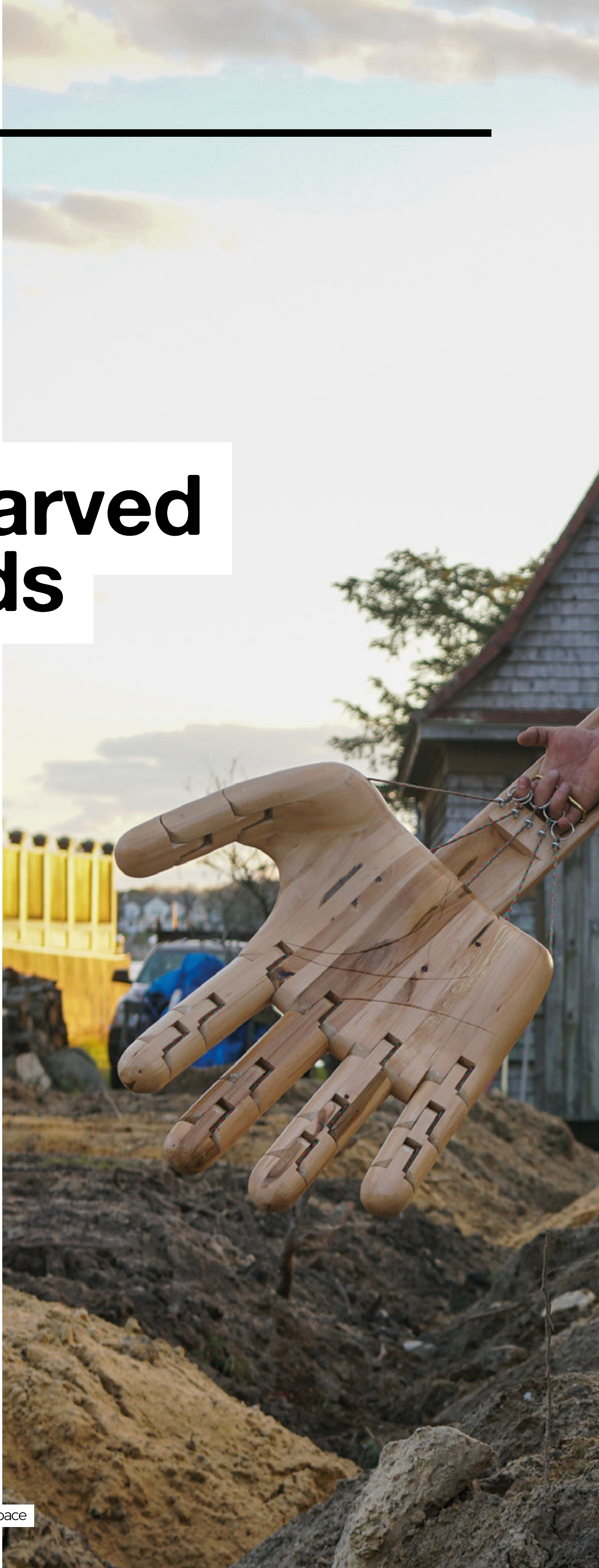
102 **Best of Breed**
The best tiny gaming platforms

108 **Direct from Shenzhen** Hot air gun
Melt solder paste with only a slight risk of electric shock

110 **Review** Bangle.js
Massive amounts of programmability on a human wrist

112 **Review** micro:bit version 2
The BBC's physical computing platform levels up

Some of the tools and techniques shown in HackSpace Magazine are dangerous unless used with skill, experience and appropriate personal protection equipment. While we attempt to guide the reader, ultimately you are responsible for your own safety and understanding the limits of yourself and your equipment. HackSpace Magazine is intended for an adult audience and some projects may be dangerous for children. Raspberry Pi (Trading) Ltd does not accept responsibility for any injuries, damage to equipment, or costs incurred from projects, tutorials or suggestions in HackSpace Magazine. Laws and regulations covering many of the topics in HackSpace Magazine are different between countries, and are always subject to change. You are responsible for understanding the requirements in your jurisdiction and ensuring that you comply with them. Some manufacturers place limits on the use of their hardware which some projects or suggestions in HackSpace Magazine may go beyond. It is your responsibility to understand the manufacturer's limits.



# Giant CNC-carved wooden hands

By **Paul Jackman**

jackmanworks.com

**One day we'll be able to hug strangers again without fear of social opprobrium.** Until that time, there's this ingenious way of maintaining social distancing made by Paul of Jackman Works. The design looks self-explanatory at first – the fingers of each hand are linked by a cord to the fingers of the wearer's hand, so each digit is exactly as moveable as your own fingers.

That simplistic view obscures the engineering that's gone into this build. Inside each finger there's a length of bungee cord acting as an extensor tendon, so that the finger will straighten out when Paul isn't actively flexing it. And the whole thing is cut out on a CNC machine, so the joints work every single time. Give him a big hand! □

**Right**
Paul makes many things (including these hands) out of wood reclaimed from old pallets

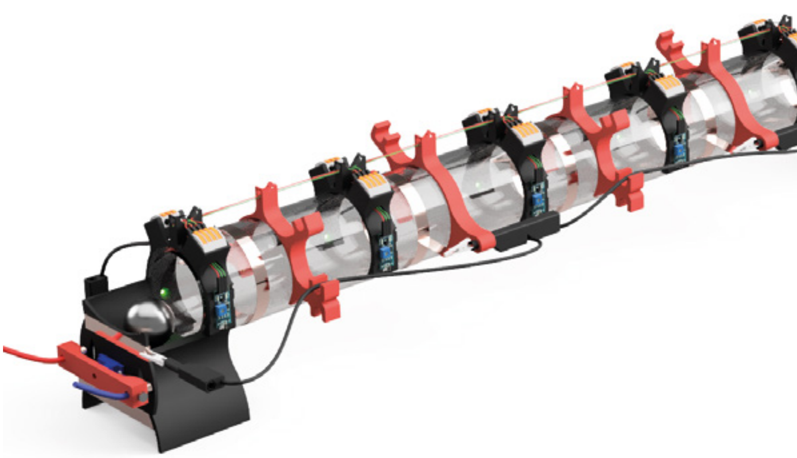
# Linear accelerator

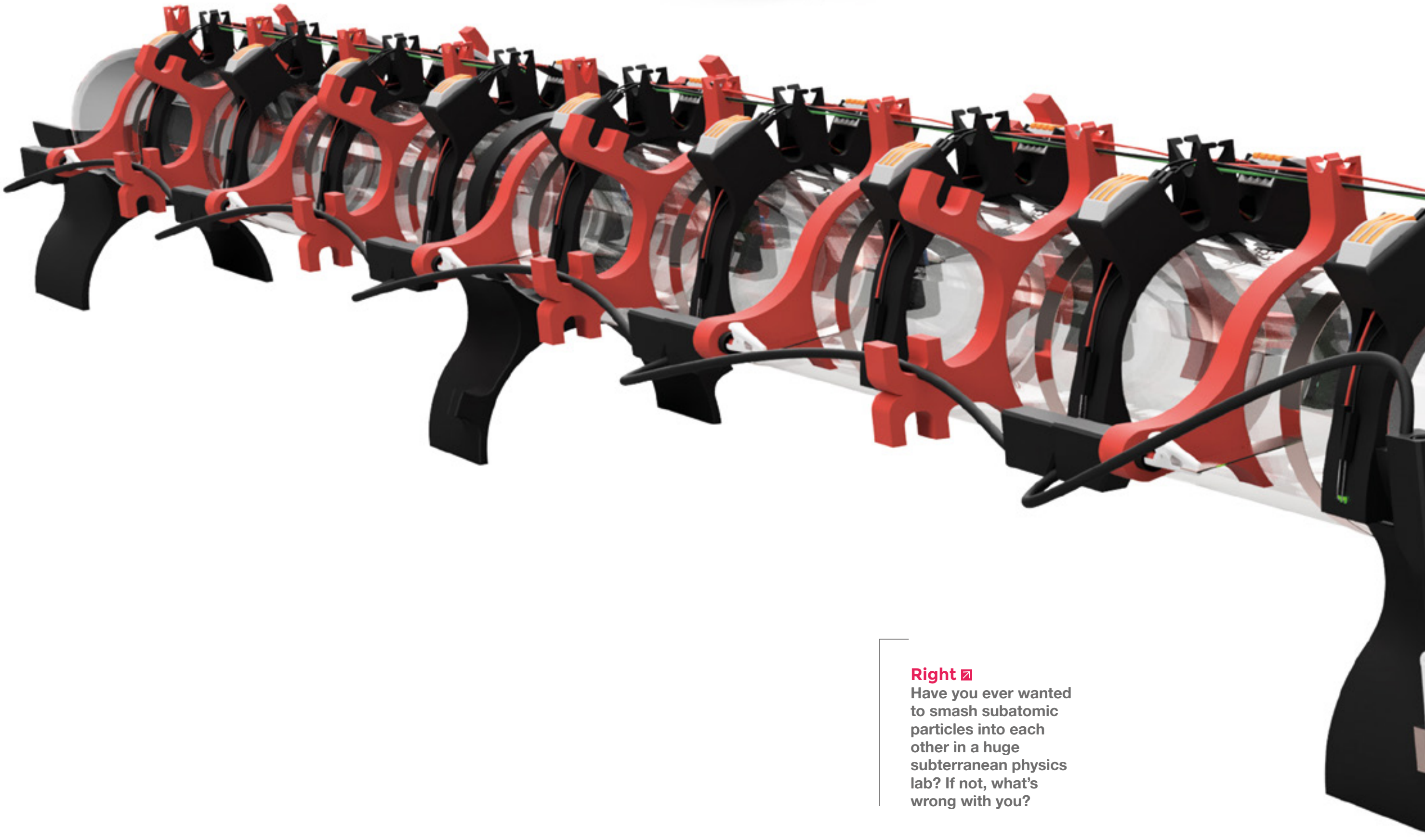
By **CERN**

hsmag.cc/Cern

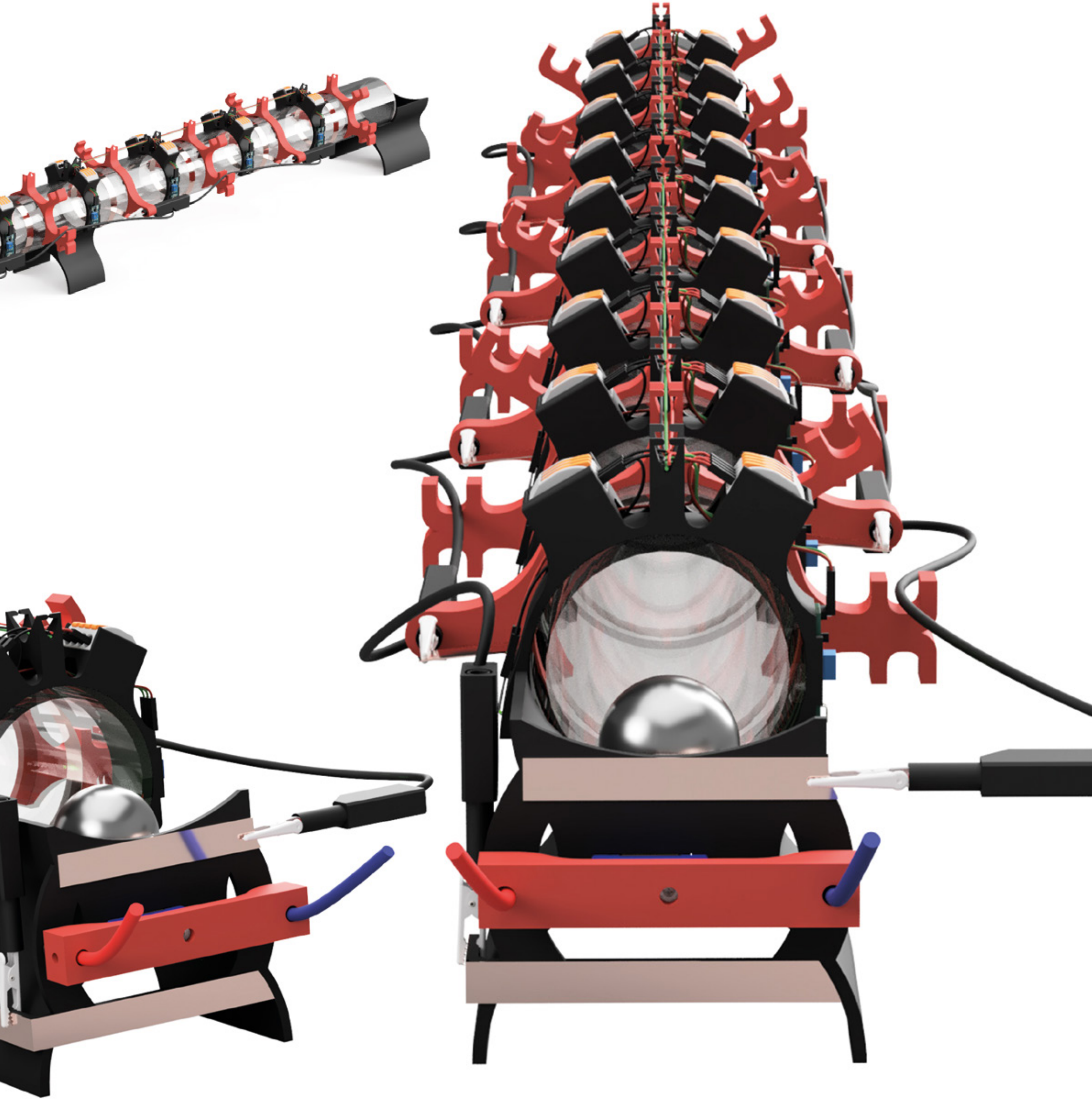
**If you've ever read Dan Brown's magisterial work *Angels and Demons*, you'll be aware of the Large Hadron Collider at CERN, the European Organisation for Nuclear Research.** In order to bring the joy of theoretical physics to as many as possible without the attendant risk of creating anti-matter and blowing up the Vatican, CERN's physics education facility has come up with this: the linear particle accelerator.

Instead of ions and protons, the particle here is a ping-pong ball coated in graphite, but the principles are the same as in the real thing – a series of electrical charges accelerates the particle down the tube. There are two versions available (CERN provides STL files and a bill of materials to build your own), one a simple on/off model, and the other controlled via an Arduino.





**Right**
Have you ever wanted to smash subatomic particles into each other in a huge subterranean physics lab? If not, what's wrong with you?



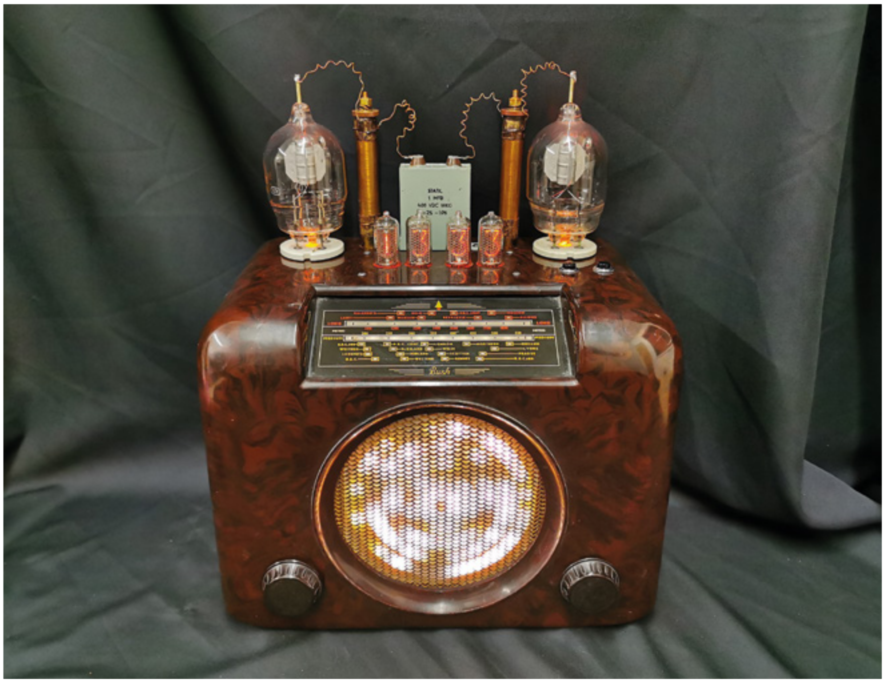
# Mesmeriser 2

By **Paul Parry**

hsmag.cc/BadDogRadio

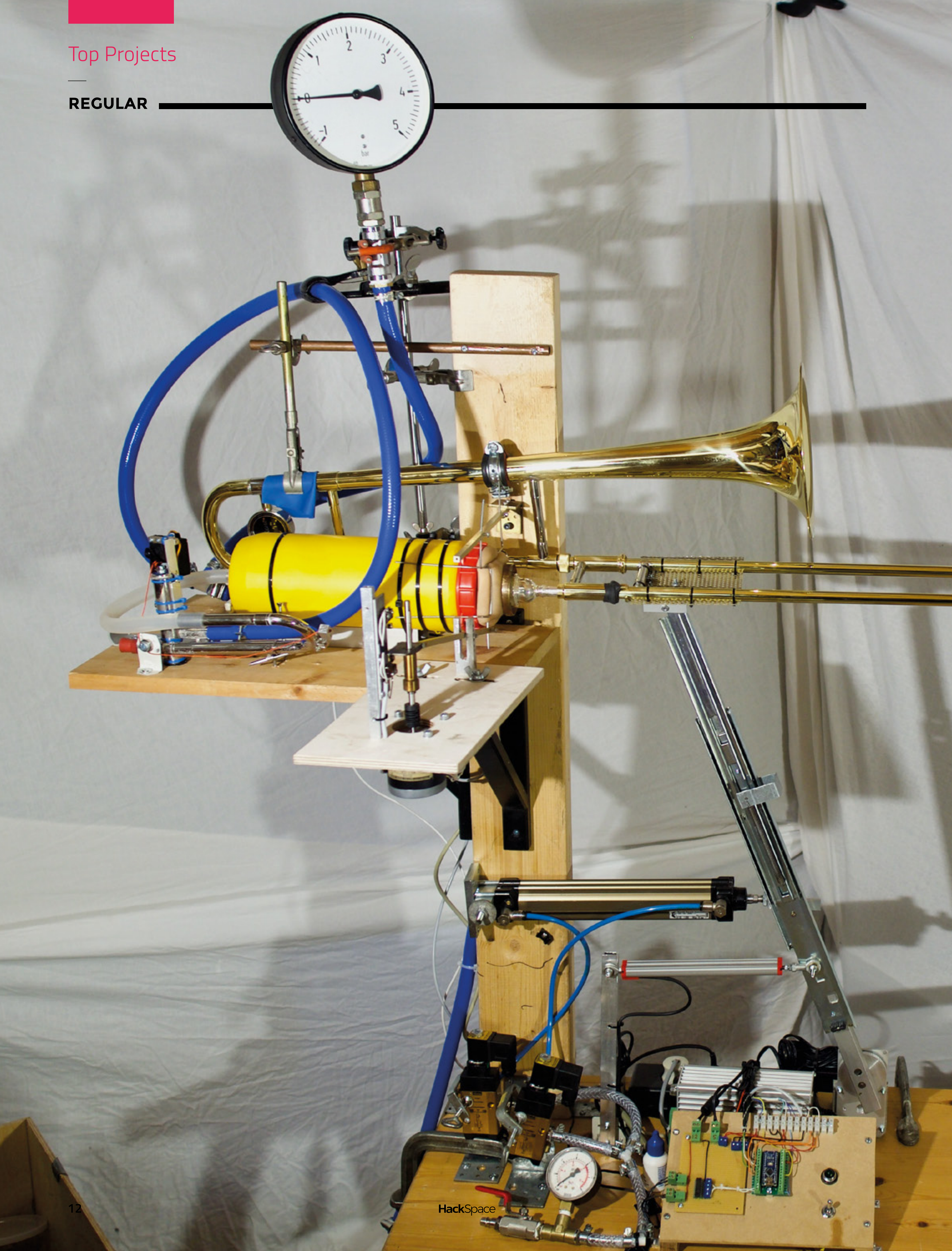
**This Bush DAC90 radio was given to Paul Parry with the request that he turn it into a Nixie clock.** Whereas it's commonplace to replace old radio internals with a Bluetooth speaker and amplifier, this time Paul removed the radio circuitry and replaced it with a series of addressable LED rings. These are diffused with some Mylar sheeting, and protected by the original mesh grille.

To preserve the original look of the radio, there's a 3D-printed light-box, complete with warm white LEDs to light up the original control panel.



**Right**
Paul took the radio's bakelite case to his local car body shop to get them to polish it up – the results are lush

STATIC
1 MFD
400 VDC WKG
2% -10%
LONG
MEDIUM
METRES
Bush



# RoboTrombo

By Isak Monrad-Aas & Thomas Hoffmann

hsmag.cc/RoboTrom

**Robots are getting above themselves.** Not content with building cars, laying undersea cables, and all manner of industrial applications, they're turning their attention to the arts as well. This robotic trombone player, by Isak Monrad-Aas and Thomas Hoffmann, comprises a pneumatic actuator, a linear potentiometer, two solenoid valves, a servo, and a stepper motor. For the moving parts, all are controlled by an Arduino Nano.

That replaces the arms and brain of a human player; an air compressor and a pair of artificial latex lips replace the human's mouth and lungs.

On this occasion, man's reach has exceeded his grasp, as it doesn't sound at all like a trombone and instead is "an over-engineered noise machine", as Isak puts it. We don't care – it's still brilliant. ☐

**Left**
The RoboTrombo was inspired in part by Martin Molin's Wintergatan musical automata

# Pico Zarch

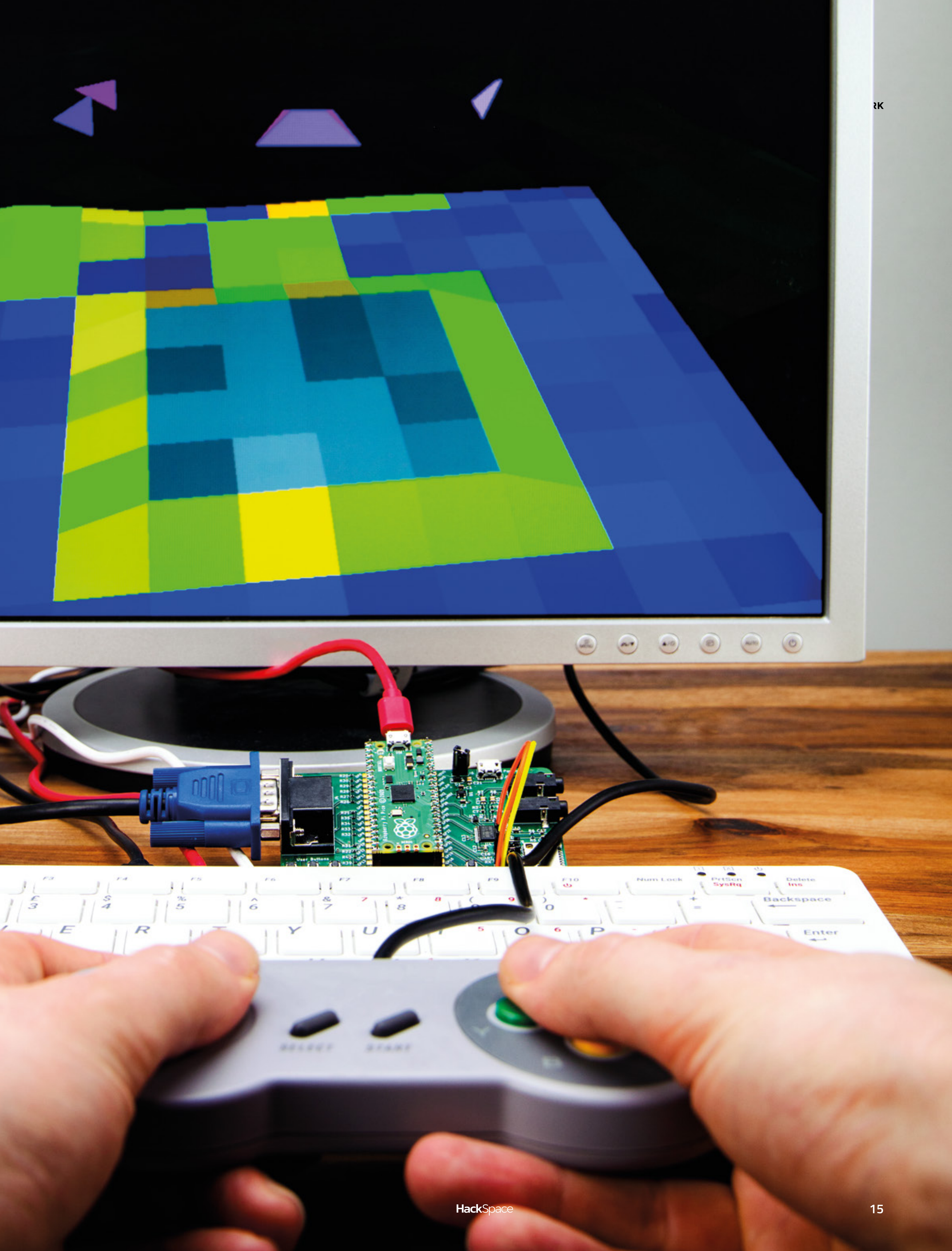
By **Eben Upton and Graham Sanderson**

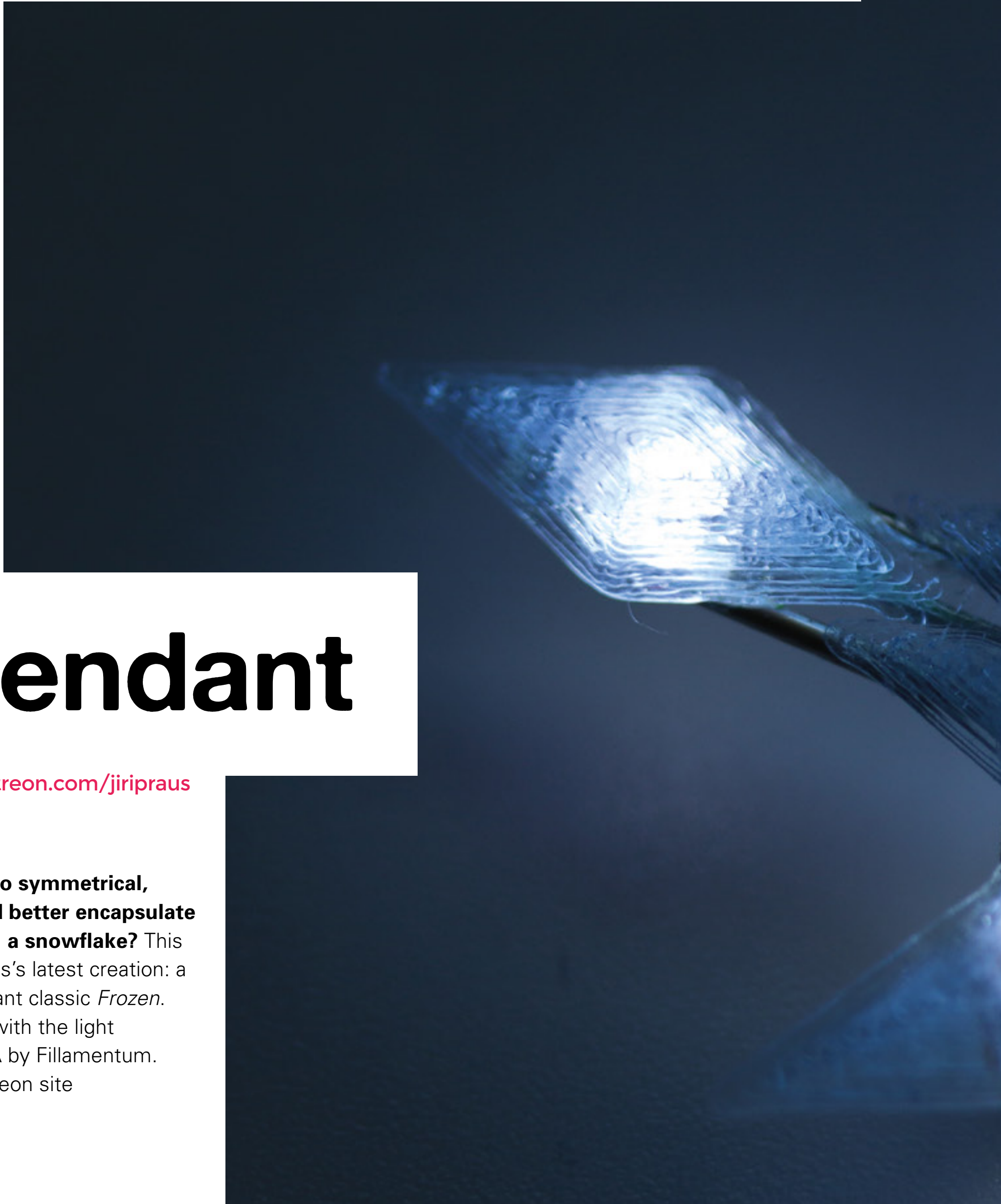
**Zarch (or Lander as its original demo was known) came out in 1987 on the Acorn Archimedes.** For the time it was a bit of a revolution – while it wasn't the first 3D game, it was probably the most sophisticated of its time. Schoolchildren across the UK (including HackSpace magazine editor Ben Everard) spent many hours of computer lessons attempting to control the fickle flying machine with the mouse.

What better game, then, to show off the power of RP2040? Eben Upton created this version (inspired by the original Zarch) to run in HTML5 (you can play it at **hsmag.cc/Ajax3D**), and it was ported to Pico by Graham Sanderson. It currently takes input via UART (so you need a computer to feed the controller data to a serial port), but everything else runs on the Pico. Output is via a VGA connector controlled by a resistor DAC. Audio is output through PWM or I2S.

There are more useful things you can do with Pico, but we're quite happy here reliving our schooldays with Zarch. □

**Right**
We'll look more at video output using Pico next issue



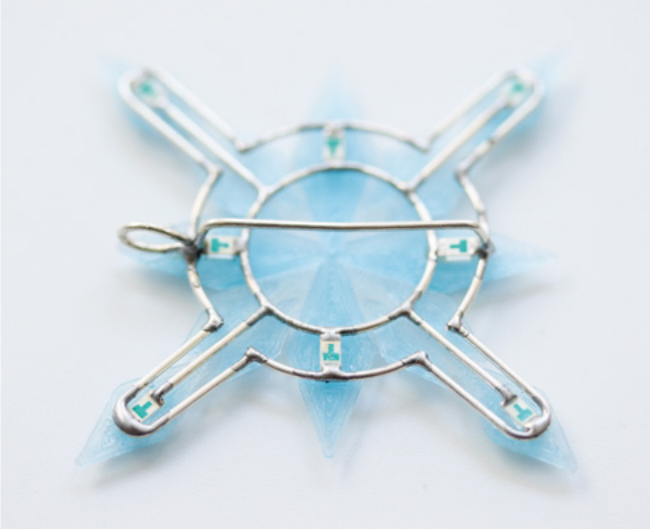


# Frozen pendant

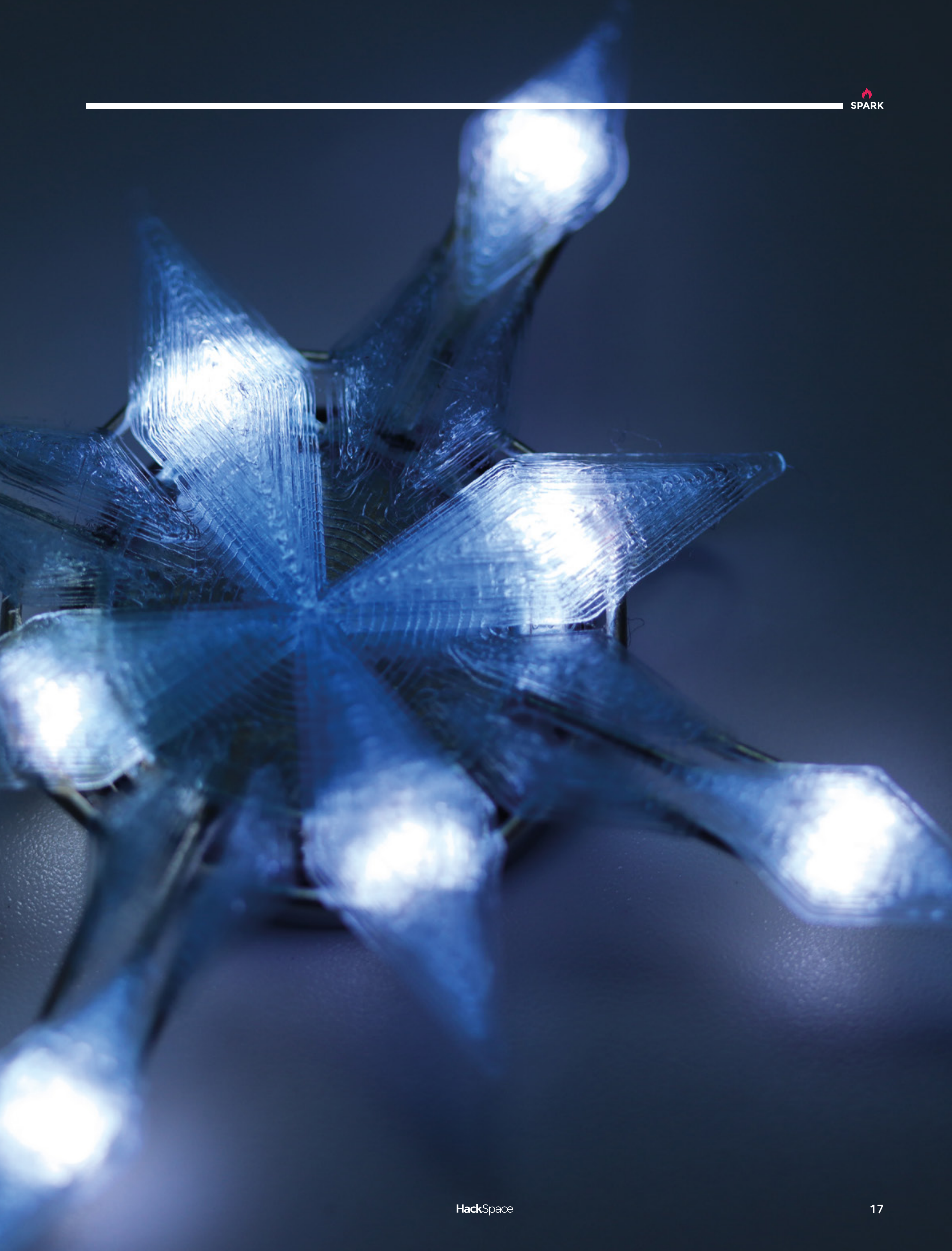
**By Jiří Praus** patreon.com/jiripraus

**Free-form soldering lends itself to symmetrical, unique designs, and what could better encapsulate symmetry and uniqueness than a snowflake?** This is Czech maker and artist Jiří Praus's latest creation: a pendant inspired by Disney's instant classic *Frozen*.

It's lit up by cold, white LEDs, with the light diffused through Iceland Blue Crystal Clear PLA by Fillamentum. Plans and instructions are available at Jiří's Patreon site **patreon.com/jiripraus**. □



**Right** ↗
"My soul is spiralling in frozen fractals all around"



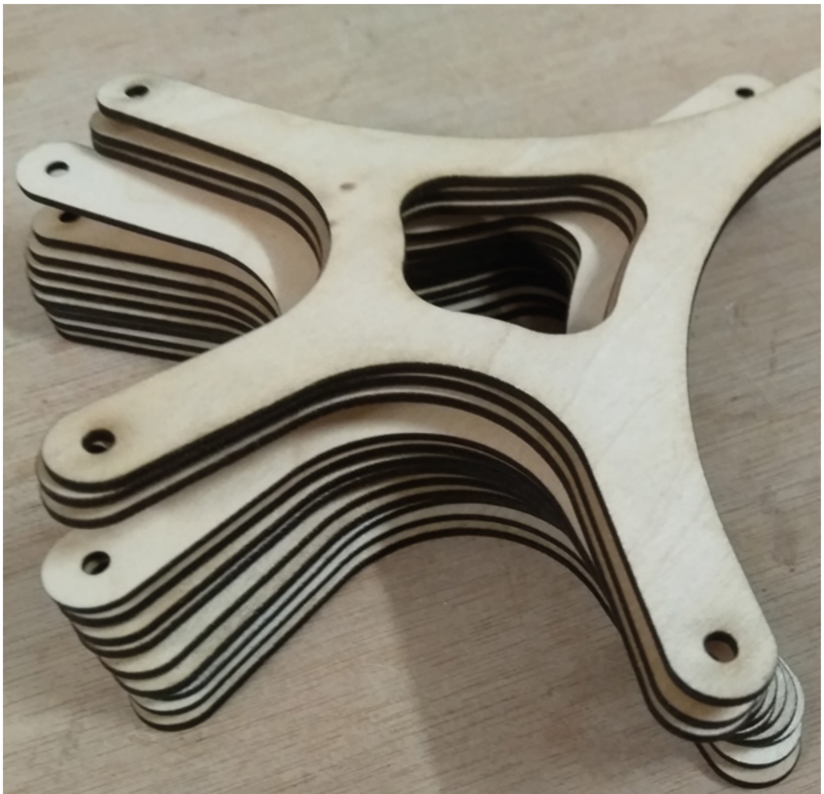
# Polyhedron light-shade

By **Victoria Joy**

hsmag.cc/LightShade

**When her local makerspace acquired a laser cutter, mechanical engineer and maker Victoria Joy knew exactly what she wanted to use it for.** This light-shade, in 1.5 mm plywood, is based on the geometry of a deltoidal hexecontahedron, a Catalan solid with 60 identical kite-shaped faces. It throws interesting shadows when lit, and doesn't look too shabby during the daytime either.

Full instructions, including a description of the design process, can be found on Instructables. Victoria explains how she modelled the polyhedron, and designed the curvy plywood shape that was then laser-cut multiple times, and assembled to form an almost spherical structure. □



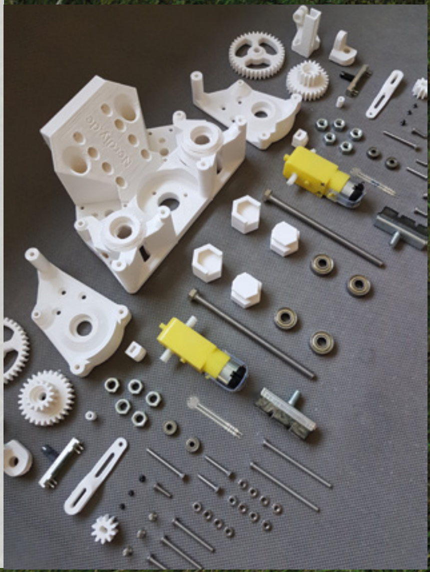
**Right**
As well as being a maker, Victoria's an environmental lawyer, wine-taster, and a founder member of Otley Maker Space

# Objet 3d'art

## 3D-printed artwork to bring more beauty into your life

**This is WinDIY.** It's an almost entirely 3D-printed HAWT, or Horizontal Axis Wind Turbine. And its creation was partly down to accident. Its creator, Fabian Steppat, happened to buy some magnets at a flea market. Rather than use them to stick notes to his fridge, he designed and built a 3D-printed disk generator, featuring 40 neodymium magnets and twelve custom-wound coils (we would normally say hand-wound, but Fabian built a machine to automate this process).

Having found himself with a generator, he then thought it would be a good idea to "harvest some free watts from mother earth", and so the seed for WinDIY was sown. We invite you to check out Fabian's full project page at **hsmag.cc/WinDIY**, where he goes through the full design considerations, but we'll leave you with the thought that the whole thing can be made on a 20×20 cm FDM printer. ◻



**Left**
**Almost every part of WinDIY is 3D printed from scratch**

# Restoration, conservation, or recycle?

## Keeping things out of landfill

**Lucy Rogers**

@DrLucyRogers

Lucy is a maker, an engineer, and a problem-solver. She is adept at bringing ideas to life. She is one of the cheerleaders for the maker industry, and is Maker-in-Chief for the Guild of Makers: **guildofmakers.org**

**Bad weather has broken a sea-wall near me and uncovered the infill – rubbish from over 60 years ago.** Glass bottles, broken plates, bricks, and animal bones now scatter the foreshore. As I kick a piece of green crockery, it reminds me of the grim Sunday dinners my nan cooked – and that I had to clean my plate before being let out into Grandad's workshop to collect the brass curls – the swarf that came off his lathe – that I would use for all sorts of decorations.

As the tide recedes, I enjoy beach-combing to see what other secrets of the past are revealed – what my grandparents' generation threw away. I froze when I thought I saw a hand grenade, but on closer inspection, it was just a mud-filled jar that once contained Peck's meat-paste. Someone with a metal detector found an old shilling and lumps of rust, orange flakes falling off like confetti, which was too far deteriorated to identify.

> **I froze when I thought I saw a hand grenade, but on closer inspection, it was just a mud-filled jar**

To get to this bit of beach, I climb up and over a man-made hill, now turned into a wildflower meadow. But a few metres below my feet is the rubbish of my generation. It's a reclaimed old refuse tip. If the sea ever breaks this open, I don't think our grandchildren are going to be so curious about what defines us – the polythene bags, nappies, and broken plastic toys that we threw away.

But maybe we have started to make a move from a throwaway society. Television seems to be full of items being repaired, restored, or recycled. I know someone who takes old AVO meters, and turns them into steampunk clocks. At first, I felt this was sacrilege, but then I realised that the hundreds he had acquired were destined for landfill. I now smile when I see old aeroplane parts turned into desks, lamps, and coffee tables. A part of our heritage is being kept – the parts reused.

I know people who use old tools – such as wooden moulding planes – with the handles worn smooth and shaped over years of use. Maybe it's not as quick as a router, but the connection it gives to those skilled workers of the past is apparently worth it.

I realise that there is a cost implication of buying things that are built to last – which is why I am delighted to see many communities have 'repair cafés' where people can donate broken tools and equipment to be fixed and resold, or they can bring in their own items for repair.

I am also excited that the circular economy – a closed-loop system where everything is reused or recycled – is gaining traction. Maybe in the future, rubbish tips will be a thing of the past. ☐

# How secure is your hardware?

## A computing platform you can trust

**Drew Fustini**

@pdp7

Drew Fustini is a hardware designer and embedded Linux developer. He is a board member of the Open Source Hardware Association and the BeagleBoard.org Foundation. Drew designs circuit boards in KiCad for OSH Park, a PCB manufacturing service, and maintains the Adafruit BeagleBone Python library.

**Regular readers of HackSpace magazine may remember that I took over this column from the iconic hardware hacker Andrew 'Bunnie' Huang.** Recently, Bunnie has been working on a really exciting new project that I wanted to share in this month's column: the Precursor. This impressive piece of open-source hardware engineering began with the question, 'how do we trust hardware?'

Software developers can use a digital signature to ensure that the bits on our computer are the same bits that are on the server, as opposed to a counterfeit version with hidden malware. It is not possible to have that same level of trust when it comes to hardware, because there are many stages between the hardware designer and the user. The Snowden files revealed the ways in which the NSA implanted chips into devices, including opening manufactured objects and adding in beacons during the distribution process (**hsmag.cc/NSA**).

There are ways to inspect and verify a piece of hardware, right the way down to the chip level, but this process is complex and expensive. Perhaps your personal threat model doesn't include being worried about the NSA (or GCHQ), but it's not just spy stuff: there are a lot of people – investigative journalists, lawyers, activists, and political or public figures – whose hardware is a real target.

This is where the Precursor comes in: an open-source hardware development platform for secure mobile computation and communication. The Precursor is a pocket-sized, lightweight device with a display, a physical keyboard, and a battery with a stand-by time measured in days (**hsmag.cc/Precursor**). In place of hardwired silicon, it is powered by an FPGA soft-core system-on-chip (SoC), allowing developers to inspect, verify, and customise pretty much every aspect of its operation. The rest of the hardware has also been designed for ease of verification, from the main PCB layout to the choice of LCD screen and keyboard.

> **The Precursor is a pocket-sized, lightweight device with a display, a physical keyboard, and a battery**

This device is part of years of research, experimentation, and thought by Bunnie and his collaborators, including Sean 'xobs' Cross. Happily, this project is now well on the way to becoming a reality after raising over $400,000 in crowdfunding orders on Crowd Supply. For the security-conscious – or security-curious – among you, Bunnie has written a number of excellent blog posts on security, open hardware, and design that are really worth seeking out (**hsmag.cc/BunnieBlog**). □

# Letters

**ATTENTION ALL MAKERS!**

If you have something you'd like to get off your chest (or even throw a word of praise in our direction) let us know at **hsmag.cc/hello**

## OOH-ER

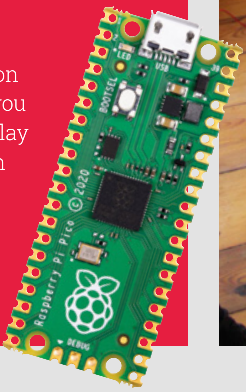
I'm sitting here watching the *Carry On* films at Christmas, Sid James chortling away, when I turn to HackSpace magazine issue 38, and a how-to on slapping jellies and touching bananas to produce music. I'm going to play around with some sound files and make my very own innuendo-laden sound effect lab. Cheers everyone!

**Will**
Boston (the one in England)

**Ben says: I was more impressed with the giant walk-on piano keyboard that Helen Leigh made in the same tutorial, but maybe that just shows the generational divide (the scene in *Big* with Tom Hanks playing the giant piano means more to me than any daft British comedy from the 1960s). Either way, I'm glad that something has fired up your creative juices!**

**GET PICO**

If you're reading this on an electronic device you won't have a Pico to play with yet – but you can get one for free with a £10 three-issue subscription. Head to **hsmag.cc/FreePico** for more details.



## DING-DONG!

There's one flaw to putting your doorbell on the internet (issue 38). It makes the wild assumption that people (e.g. delivery drivers) will actually press the button, and not just throw a 'sorry we missed you' card through the letterbox and run away. What it really needs is some sort of laser trip-wire that will alert me when people get close, or a microphone to pick up on the old-fashioned visitors who physically knock on the door.

**Brian**
Dublin

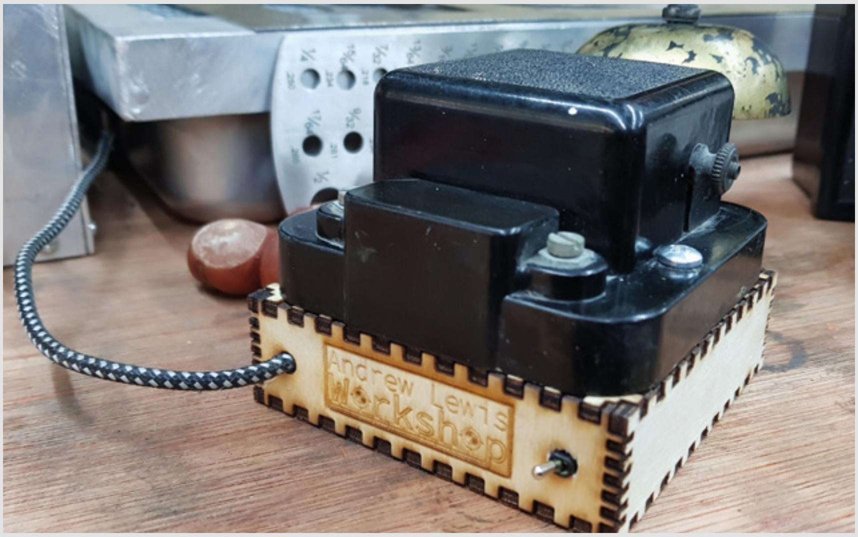
**Ben says: In the interests of fairness, Andrew Lewis had to assume goodwill from the users of that project. Of course, there are always people who abuse the rules around package deliveries (Mark Rober and his glitter bombs are dealing with them one by one), but we live in hope, and must assume that most people are good. In any other way lies madness.**

## MOVIES

Ever since the day my auntie decided I would be too young to understand the plot of *Back to the Future*, and took me to see *The Care Bears Movie* instead, I've always wanted my own time machine – do all the wiring, lights, flux capacitor, make a little Mr. Fusion nuclear reactor, and some time circuits, just like the makers in your latest issue. In fact, I've kept hold of Jérôme Montignies's time circuit clock from HackSpace magazine issue 37, with the intention of making one myself over a period of enforced idleness. But there's one last part that's eluding me. I just can't seem to find a DeLorean to put it all in for less than £25,000, which kind of takes it out of weekend project territory, budget-wise. Do you reckon it would work in a Citroën 2CV?

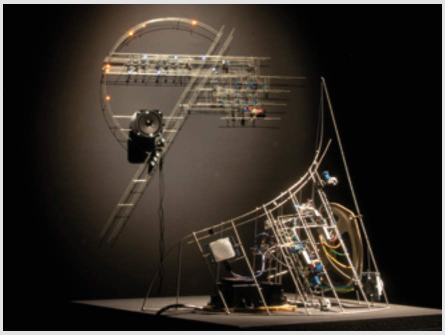
**Mark**
Carcassonne

**Ben says: Do it. Do it now.**



## APOLOGY

Last issue, we featured Eirik Brandal's Waldian sculpture, winner of the Hackaday.io circuit sculpture contest, we didn't credit the image creator though – that was Miha Fras. Sorry about that.



# CROWDFUNDING NOW

## Eduponics Mini

Programmable plant care

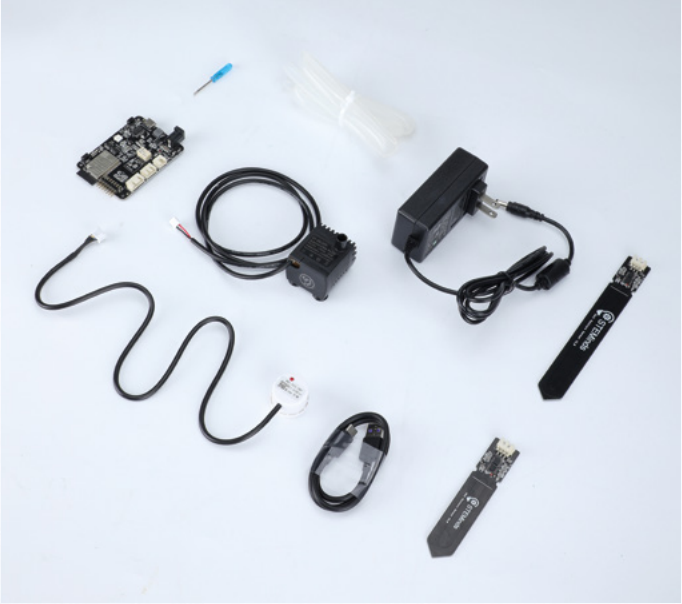
crowdsupply.com

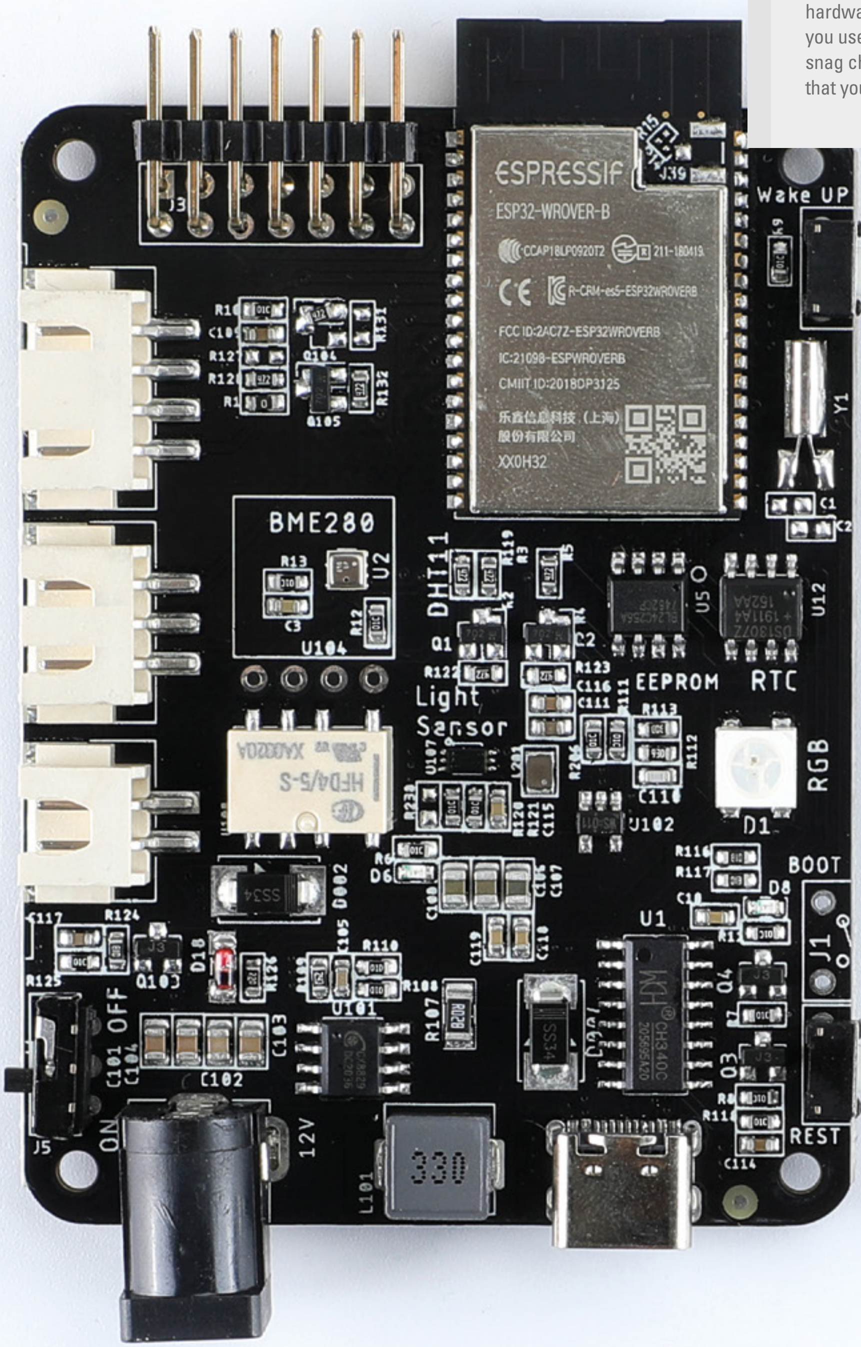
**Plants are great things to have around.** They look nice, they convert carbon dioxide into oxygen, and sometimes you can even eat them. However, they do take a bit of looking after. Eduponics Mini is an ESP32-based kit designed to make caring for these fickle beings a little easier.

The control board includes the microcontroller and sensors for light, temperature, pressure, and humidity. It's also got connectors to add in soil moisture sensors, and – crucially – a pump so you can get it to water your plants automatically.

There's an accompanying smartphone app which lets you keep control of your garden when you're on the go, but the controller board runs MicroPython, so you can program it to run however you like.

We've not tested one out, but it looks like it could be a great option for keeping your houseplants alive, or even getting your greenhouse running smoothly. It looks perfect for the hacker who's more used to programming than tending plants. ☐





## BUYER BEWARE

When backing a crowdfunding campaign, you are not purchasing a finished product, but supporting a project working on something new. There is a very real chance that the product will never ship and you'll lose your money. It's a great way to support projects you like and get some cheap hardware in the process, but if you use it purely as a chance to snag cheap stuff, you may find that you get burned.

# Build a Makerspace for Young People

Join our **free online training course on makerspace design** to get expert advice for setting up a makerspace in your school or community.

**Sign up today: rpf.io/makerspace**

Raspberry Pi

The Raspberry Pi Foundation. UK registered charity 1129409

#RPiLearn

HackSpace
TECHNOLOGY IN YOUR HANDS

# LENS

HACK | MAKE | BUILD | CREATE

Uncover the technology that's powering the future

PG 48

## HOW I MADE OPTICAL SOUND DECODER

Long-lost voices come back to life with this ingenious optical sound extractor

PG 54

## INTERVIEW: EBEN UPTON

Why on earth would you spend £3m developing a chip, only to give it away on the front of a magazine?

PG 62

## IMPROVISER'S TOOLBOX: MAGNETS

Harness the power of mysterious forces more constructively than sticking things to the fridge

PG 30

## RASPBERRY PI PICO

LOW-COST • HIGH-PERFORMANCE • FLEXIBLE I/O

Raspberry Pi Pico © 2020
BOOTSEL
LED
USB
DEBUG

# RASPBERRY PI PICO

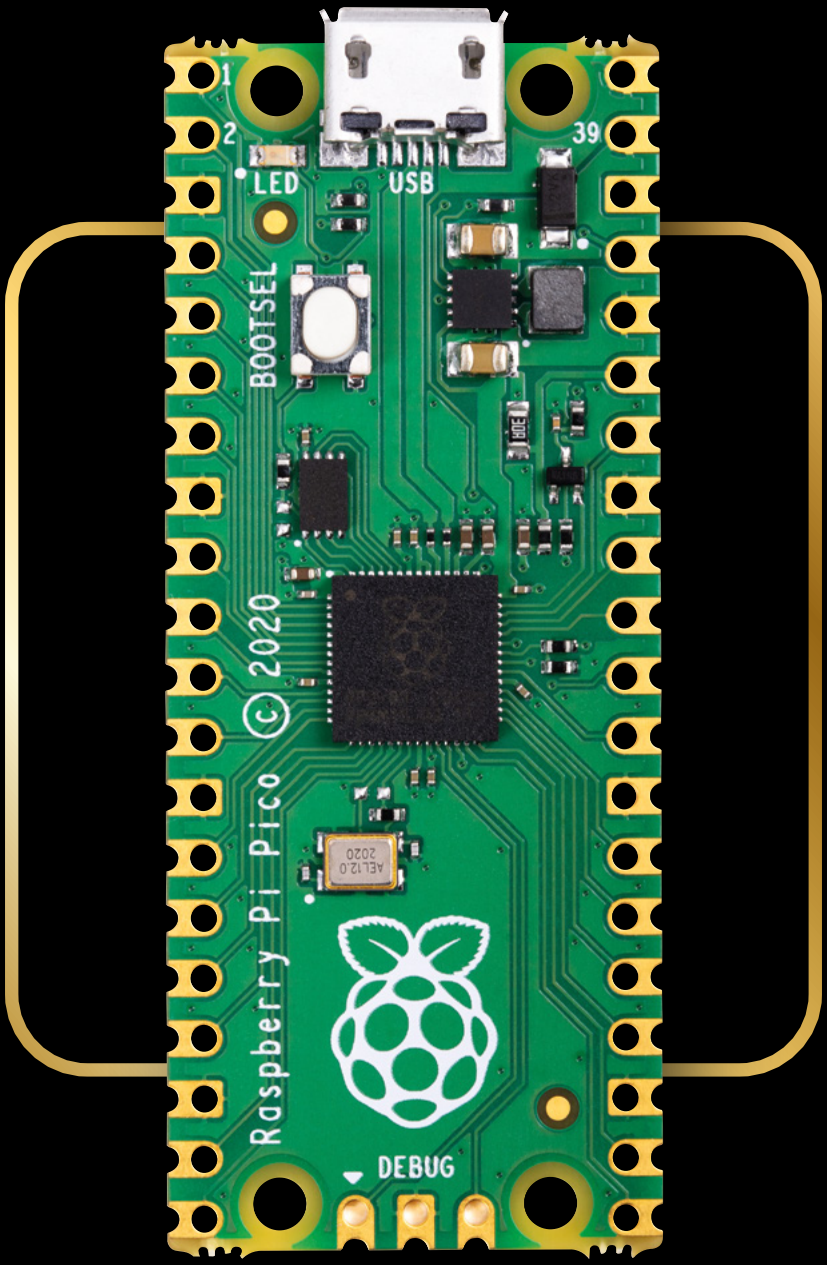
LOW-COST • HIGH-PERFORMANCE • FLEXIBLE I/O

**If you're reading the paper version of this magazine, you have in your hands right now a brand new microcontroller board.** If you're reading the digital version, you can get a free Pico when you get a £10 three issue subscription. Head to **hsmag.cc/FreePico** for more details. Alternatively, you can buy one for $4 plus local tax and shipping from your favourite electronics retailer.

Raspberry Pi Pico is designed to be high-performance, low-cost, and provide a flexible way of interacting with other hardware. With two ARM Cortex M0+ cores running at up to 133MHz and 264kB of RAM there's plenty of processing power for most tasks. The 26 GPIO pins include two I2C, two SPI, two UART, three analogue inputs, and a new feature called Programmable I/O (PIO) which we look at in detail on page 40. There's also 2Mb of flash for your programs and data.

Using these features, Pico could be the brains behind your next robot, DIY smart gadget, games system, electronic musical instrument, and more.

We'll look at all these features of the chip, but no doubt you're itching to start using it, so let's dive straight in and have a play. Turn the page to learn how to connect to MicroPython. →


1
2
39
LED
USB
BOOTSEL
Raspberry Pi Pico
© 2020
DEBUG

# MICROPYTHON

## DIVE STRAIGHT IN

**We'll take a detailed look at the features of Pico in the next few pages, but let's start by diving right in and coding.** There are currently two ways of programming your Pico: MicroPython and C/C++. Let's take a look at MicroPython here as it's the quickest and easiest way to get started. If you'd rather get started with C, take a look at **rptl.io/rp2040**.

First, you need to set your Pico up with the firmware. Head to **rptl.io/pico** and download a UF2 file with the latest version of MicroPython. Then, unplug your Pico (if it's already plugged in), press the BOOTSEL button, plug it in to your computer, and release the button. You should now see Pico appear as a USB mass storage device. Drag and drop the UF2 file onto the device and it will disappear. Your Pico is now running MicroPython. You can repeat this process to get MicroPython back if you program it with different firmware (or a C/C++ program).

As well as your Pico, you need a little bit of software on your computer. You can use whatever text editor and serial monitor you like, but for ease of use we recommend Thonny, at least for getting started. You can download this from **thonny.org**. It's available for most major platforms including Windows, macOS, Linux, and Raspberry Pi.

Once Thonny is installed, open it. You need to tell it how to connect to your Pico, so go to Run > Select Interpreter. You should pick 'MicroPython (RaspberryPi Pico)' in the first box. If this isn't available, then you're using an older version of Thonny – if possible, you should update this, but if not, you can use 'MicroPython (Generic)'. After selecting the version, you need to select the serial port your Pico's connected to. Depending on what else you have connected to your computer, you may only have one option there. If there's more than one, it may take a little trial and error to find the right one. You can always unplug Pico to see which port disappears from the list, then plug it back in. When it's connected properly, you'll see something like the following in the Shell section of the main interface:

```
MicroPython v1.13 on 2020-10-14; Raspberry Pi
Pico with cortex-m0plus
```

### STAYING SAFE

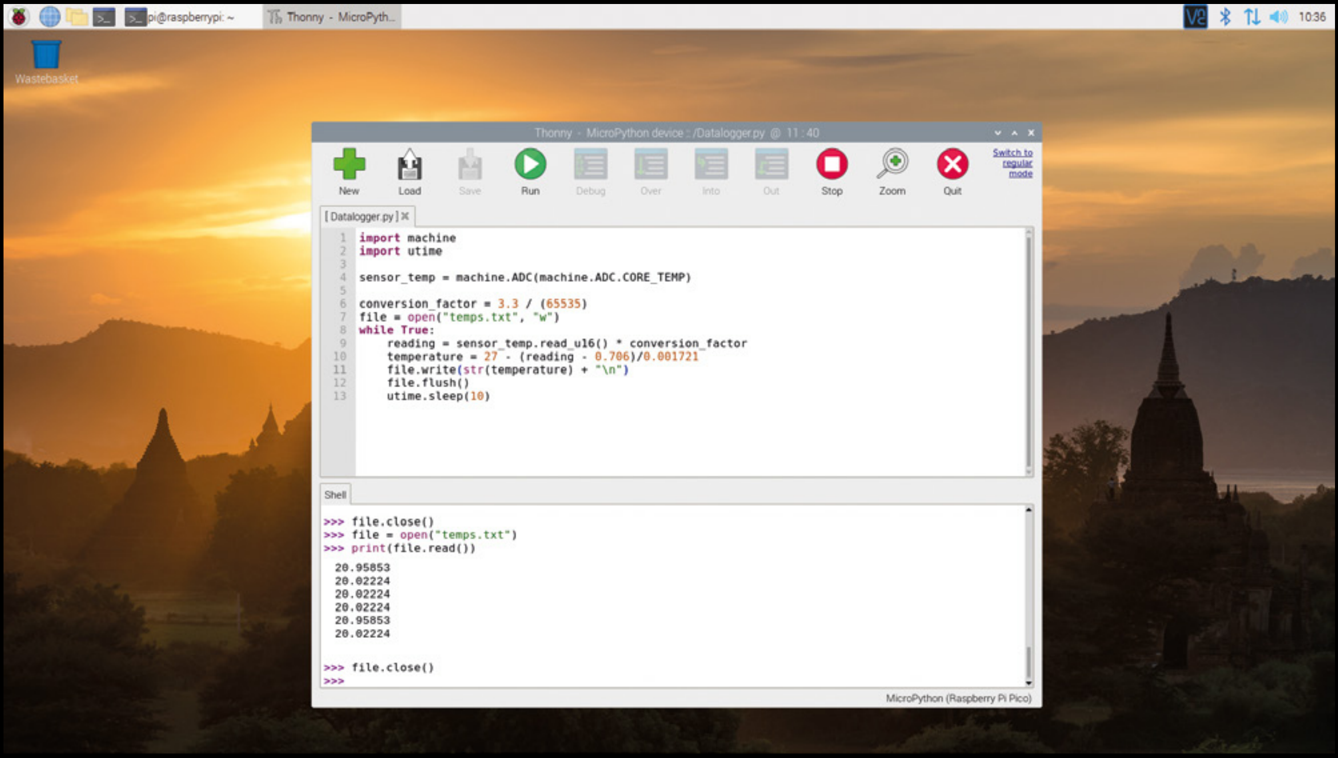
The two most likely ways of damaging your Pico are putting too much voltage into it, and drawing too much current from an input/output pin. Pico works on 3.3V, so you should never put more than this into one of the pins. Applying a lower voltage to a pin won't damage it, but may mean that it doesn't correctly register when a pin is high. If you need to read a lower voltage, you could use an analogue input.

The maximum recommended current draw is 12mA. If you're unsure of whether a component you connect will draw more than this, you can add a resistor of at least 275 ohms. This may reduce the current to the point the component won't work, but it will mean that you don't damage your Pico.

If you need to drive a component that needs more than this (such as a motor), you'll need a circuit that uses your low current GPIO pin to switch a high-powered circuit. This can be as simple as a transistor, or you can get modules for driving high-current devices such as motors.

39
USB
LED
BOOTSEL
1
2

**Above** There's a green LED on pin 25, which is a great way of testing out that everything's working


Thonny - MicroPython device :: /Datalogger.py @ 11 : 40
New
Load
Save
Run
Debug
Over
Into
Out
Stop
Zoom
Quit
Switch to regular mode
[ Datalogger.py ]
Shell
MicroPython (Raspberry Pi Pico)

```
import machine
import utime

sensor_temp = machine.ADC(machine.ADC.CORE_TEMP)

conversion_factor = 3.3 / (65535)
file = open("temps.txt", "w")
while True:
    reading = sensor_temp.read_u16() * conversion_factor
    temperature = 27 - (reading - 0.706)/0.001721
    file.write(str(temperature) + "\n")
    file.flush()
    utime.sleep(10)
```

```
>>> file.close()
>>> file = open("temps.txt")
>>> print(file.read())
 20.95853
 20.02224
 20.02224
 20.02224
 20.95853
 20.02224


>>> file.close()
>>>
```

> “ THERE ARE TWO DIFFERENT WAYS OF RUNNING MICROPYTHON CODE ”

There are two different ways of running MicroPython code. You can type it in the interpreter (at the bottom of the window where lines start with >>>). Here, each line is executed as soon as you press **ENTER**. For example, if you enter `print("hello world")`, you’ll immediately see ‘hello world’ come back. The other is in the text editor. Here, you write a script and, when you’re ready, you run the whole script in one go. For example, enter `print("hello world")` here and nothing should happen. If you press the green arrow in the toolbar (or **F5** on your keyboard), you’ll be prompted to save the file (this author prefers to save them on his computer rather than the Pico as it’s easier to keep track of them there), then you’ll see the following in the interpreter:

```
>>> %Run -c $EDITOR_CONTENT
hello world
```

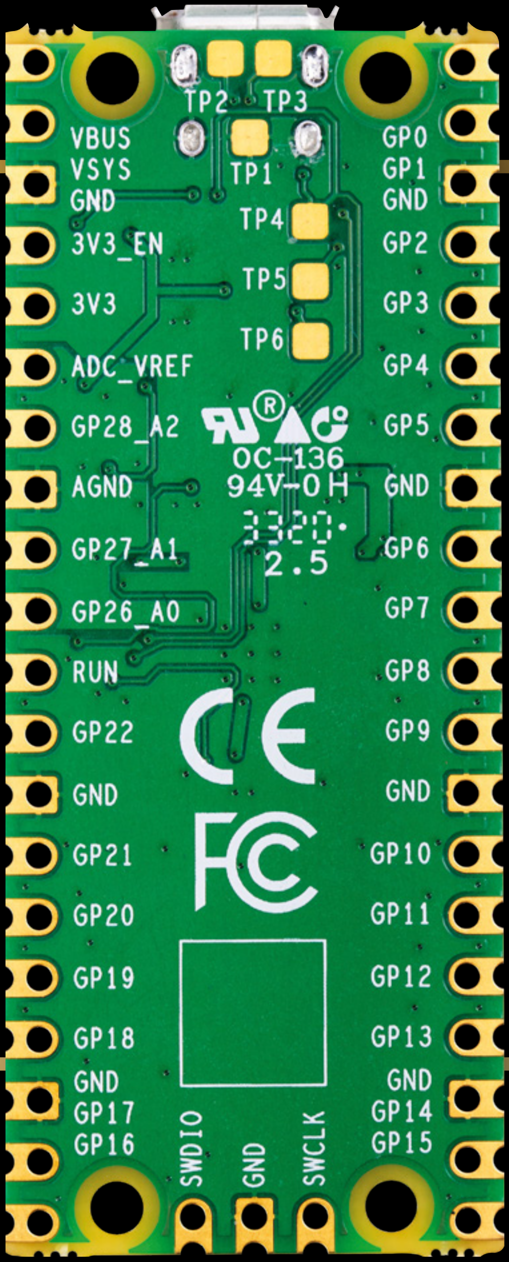
Interactive mode is great for quickly trying something out when you don’t know the exact syntax or how to do something.

Now we’ve got our development environment set up and running, let’s try a simple program. As is traditional with microcontrollers, let’s make the LED flash on and off. →

**Above** ◈
**The Thonny interface is quite stripped back, but provides quick and easy access to the basic features**

## RUNNING PERMANENTLY

Hitting the Run button to set your script going is great for testing things out, but when you’re running a project, you don’t want to have to connect your Pico to a computer and run the file. Fortunately, there’s a way around this. If you save your file on the microcontroller as **main.py**, it will automatically run each time you power on the device.


VBUS
VSYS
GND
3V3_EN
3V3
ADC_VREF
GP28_A2
AGND
GP27_A1
GP26_A0
RUN
GP22
GND
GP21
GP20
GP19
GP18
GND
GP17
GP16
TP2
TP3
TP1
TP4
TP5
TP6
OC-136
94V-0 H
3320
2.5
SWDIO
GND
SWCLK
GP0
GP1
GND
GP2
GP3
GP4
GP5
GND
GP6
GP7
GP8
GP9
GND
GP10
GP11
GP12
GP13
GND
GP14
GP15

**Above** ◈ You can see the pin numbers on the bottom of the board

> “ I’D REALLY LIKE TO SEE PEOPLE MAKE LITTLE GAMES CONSOLES ”

## MEET THE TEAM

We spoke to some of the people behind the Raspberry Pi Pico. Look out for these boxes to hear about the board from the folks who made it:

**HackSpace: What are you excited to see people do with Pico?**

**Luke Wren, Hardware and software:** There’s a guy I’ve followed on YouTube for a while called CNLohr and he’s done some really cool stuff with other microcontrollers – things like bit-banging NTSC video – and I think PIO is just made for people like that. They’re going to have a lot of fun bit-banging things you wouldn’t ever expect to see on a microcontroller. I’m excited to see what people like that do with PIO once they work out what it can do and the things you can connect it to.

**HS: Do you have some specifics in mind?**

I’m pretty sure you can bit-bang 10 Base-T Ethernet which is cool for a connected sensor. There’s also been this undercurrent on this chip of doing SNES-style graphics without building in any specific graphics hardware. I’d really like to see people make little games consoles and things like that.

```
import machine
import time

pin = machine.Pin(25, machine.Pin.OUT)

while True:
    pin.value(1)
    time.sleep(1)
    pin.value(0)
    time.sleep(1)
```

This will continue to run forever unless you press **CTRL+C** to stop it.

If you're unfamiliar with Python, one of its more unusual features is that whitespace is important, so you need to have a consistent indent for the lines after `while True:`.

This program first imports a pair of modules that it'll need: `machine` brings in the important bits for accessing the underlying hardware, and `time` brings in the sleep method that we need to pause the loop.

First, we have to set the function of the pin; here, `machine.Pin.OUT` has set it to output. If we wanted to read values in, we could use `machine.Pin.IN`. You can also set more advanced functions such as pull-ups here.

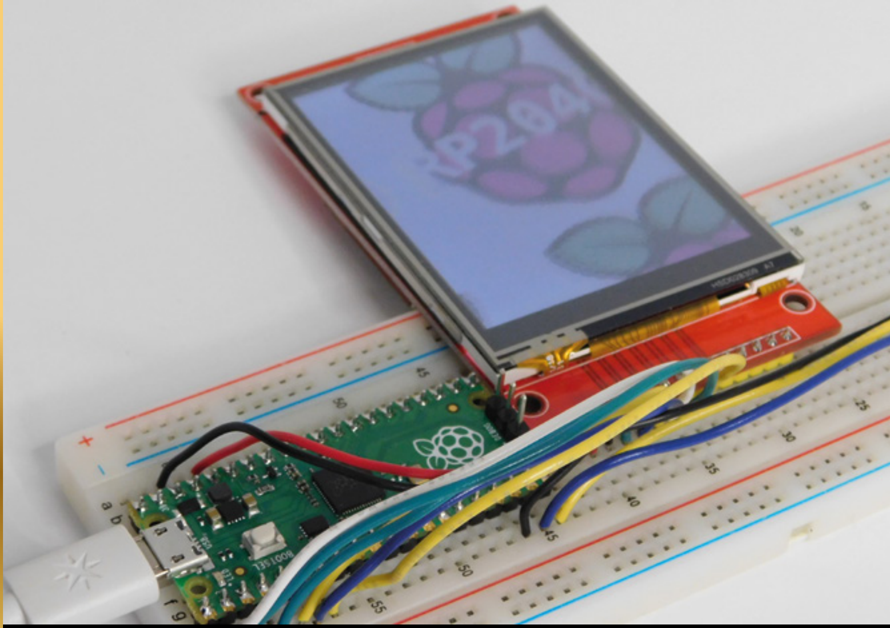
That's the basics of running MicroPython on Pico. If you're familiar with Python or MicroPython already, then you may be happy to dive right in and start programming. You can find the core docs at **docs.micropython.org/en/latest**.

If you're new to this programming language, then you can get the brand new book *Get Started with MicroPython on Raspberry Pi Pico*. Buy it online or download it for free from **hsmag.cc/MPBook**.

If you'd just like to take a look at some example code, you can see the examples from the book at: **hsmag.cc/dfmGtJ.**

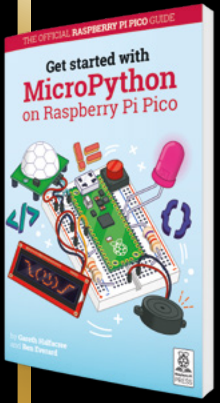
There are also some examples showing off the unique features of Pico at: **hsmag.cc/Ly7wV0**.

Turn the page and we'll take a look at what the different parts of your Pico are, and how they work together to make the microcontroller board run. →



**Above**
After soldering male pins to Pico, you can put it on a breadboard for easy connection to electronic components

**Below**
This official guide book will help you to start programming your Pico



## RP2040-SPECIFIC FEATURES

Most of MicroPython is standard, and the code that runs on a Pico will also run on a PyBoard or other MicroPython-compatible microcontroller. However, all microcontrollers have different functionality. This is expressed through board-specific modules.

For RP2040, this means the RP2 module. In here you'll find a few bits, but perhaps the most useful are the sections for accessing Programmable IO. This lets you program state machines that run independent of the CPU and handle data transfer protocols through the GPIO pins. See page 40 for more information.

There are some examples of using these in the MicroPython Examples GitHub repository at **hsmag.cc/qXiV0f**. There are full details on how to use these in the Pico Python SDK book available at **rptl.io/pico**.

> “ THERE'S ALSO 2MB OF FLASH STORAGE ON THE PICO BOARD ”

## WHAT IS A MICROCONTROLLER?

Pico is based on the RP2040 microcontroller. This microcontroller is like a stripped-down computer. It's got two processing cores and some RAM. There's also some flash storage on the Pico board (see overleaf for full specs). The main difference between a microcontroller and a normal computer is that the microcontroller isn't designed to be a complete platform in itself. You program it from a normal computer (you can use Windows, macOS, or Linux – such as Raspberry Pi Model A or B), and then the microcontroller runs this program.

On this page, we'll guide you through connecting your Pico to your computer and getting a programming environment (MicroPython) set up so you can start programming your Pico.

# RASPBERRY PI PICO

## WHAT'S ON YOUR BRAND NEW MICROCONTROLLER BOARD?

**1** The microcontroller at the heart of Pico is a brand new chip designed by Raspberry Pi. It's high-performance, low-cost, and has a host of flexible interfacing options. We look at all this on the next page.

**2** This oscillator helps your microcontroller keep ticking along at the right speed.

**3** Alongside the RP2040 chip, your Pico also includes 2MB of flash memory. If you're used to the gigabytes of memory you get in modern computers, this may not sound like much, but microcontroller programs don't tend to be very large. If you do need to store lots of data (such as if you're logging sensor values), you could add additional storage such as an SD card.

**4** The micro USB port on the front of Pico can be used to supply power and to send programs and data to the board. It also has more advanced functions. Your Pico can act as a USB device and interact with a computer in a similar way to any other USB device such as a keyboard or mouse. Want to build a custom game controller? Pico can do that.

A little unusually, Pico can also act as a USB host. This means you can plug other USB devices into it. You can use a regular USB keyboard or mouse to get data in. In principle, any USB device should work, but bear in mind that your program will need to understand how to communicate with the device.

There are some examples of both device and host USB at **hsmag.cc/kVFRIG**.

> " PICO CAN ALSO ACT AS A USB HOST "

**5** You can supply voltage either through the USB port or VSYS. If you attach a power supply (such as a battery) to VSYS via a Schottky diode, the system will take the larger of the two voltages. This allows you to have a backup power supply, or a separate power supply and not need to worry about problems when plugging Pico in for programming.

Pico can take a wide range of input voltages: from 1.8 V to 5.5 V. You can power it from a single lithium-ion cell, three AA batteries, or any other power source in the range. There are more details on powering Pico, and example circuits, in the Pico data sheet at **rptl.io/pico**.

**6** You can turn the 26 GPIO pins on and off, or read whether or not they have voltage applied to them from your code. They can turn LEDs on and off, read button presses, and control all manner of other hardware. If you need to interface with other digital devices, there are two each of I2C, SPI, and UART. There's also a flexible I/O controller called PIO that makes it easy to connect to a wide range of devices (see page 40 ).

**7** GPIO pins 26, 27, and 28 can all read analogue values. They return a 12-bit number that shows the voltage (between 0 and 3.3 V). If you'd rather use a different maximum voltage (up to 3.3 V max), you can set it via the ADC_VREF pin.

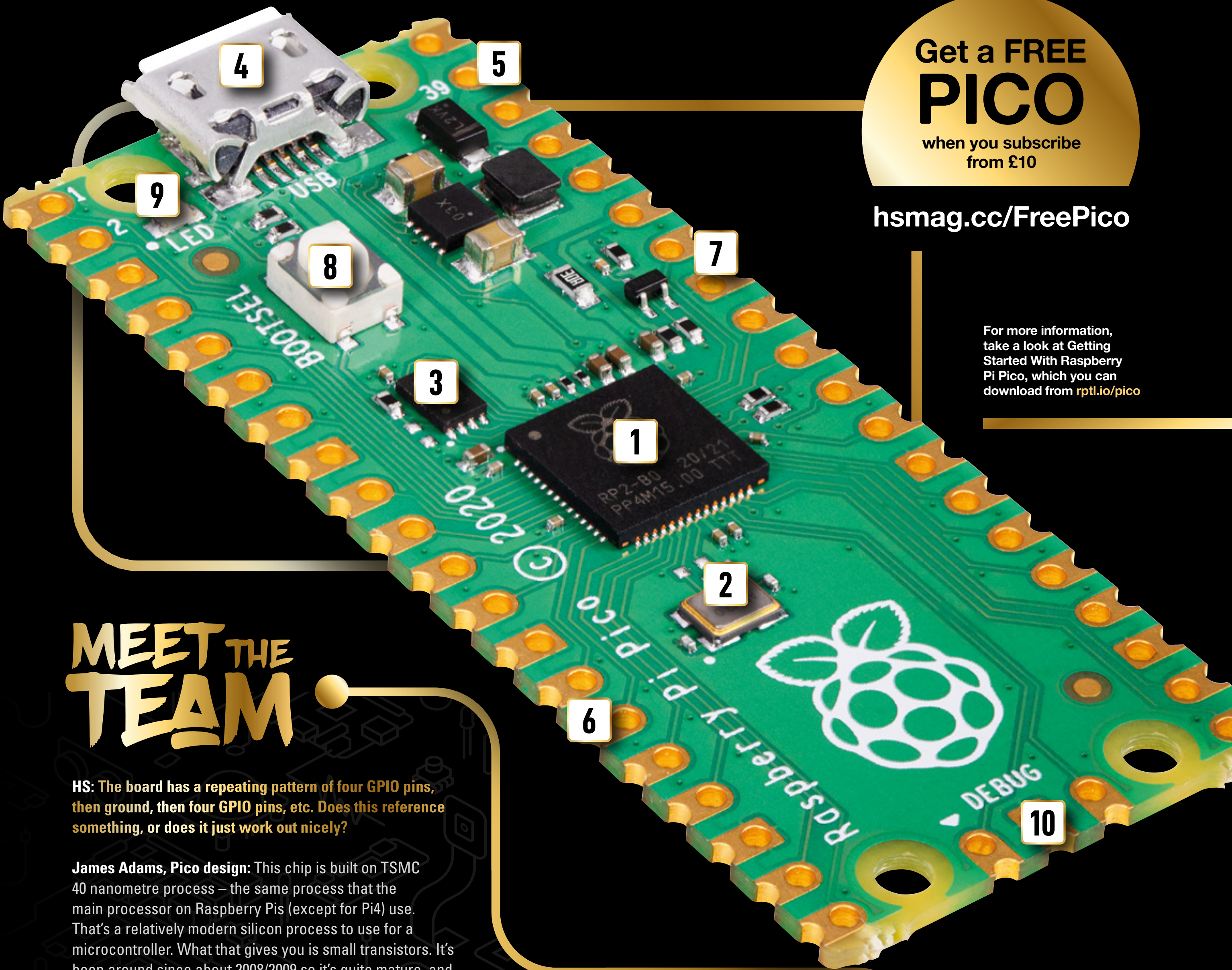
**8** The button on Pico is used to enter programming mode. Unplug Pico, hold down this button, then plug it in, and it'll show up as a USB mass storage device that you can copy UF2 files to.

You can get input from this button in your C/C++ programs – take a look at this example for details on how: **hsmag.cc/xbYOhc.**

**9** GPIO 25 is connected to a green LED that you can program however you like. It can be an indicator that something's happening on your project, or even a Morse code output.

**10** Debugging microcontrollers can be tricky. When something's not working, it's not always easy to work out what's not working, let alone why. Fortunately, Pico exposes three pins on the bottom edge, and these are for debugging using the Serial Wire Debug (SWD) protocol. If you're using a Raspberry Pi as your main device to program Pico, you can connect these directly to the GPIO pins with no extra hardware other than a few wires. If you're using a different computer, you'll need an extra adapter. Pico can be used as a debug adapter for another Pico using the Picoprobe program.

This interface allows you to add up to four breakpoints and two watchpoints. This gives you much more control about how your program runs as you develop software. →



Get a FREE
PICO
when you subscribe
from £10

hsmag.cc/FreePico

For more information, take a look at Getting Started With Raspberry Pi Pico, which you can download from rptl.io/pico

## MEET THE TEAM

**HS: The board has a repeating pattern of four GPIO pins, then ground, then four GPIO pins, etc. Does this reference something, or does it just work out nicely?**

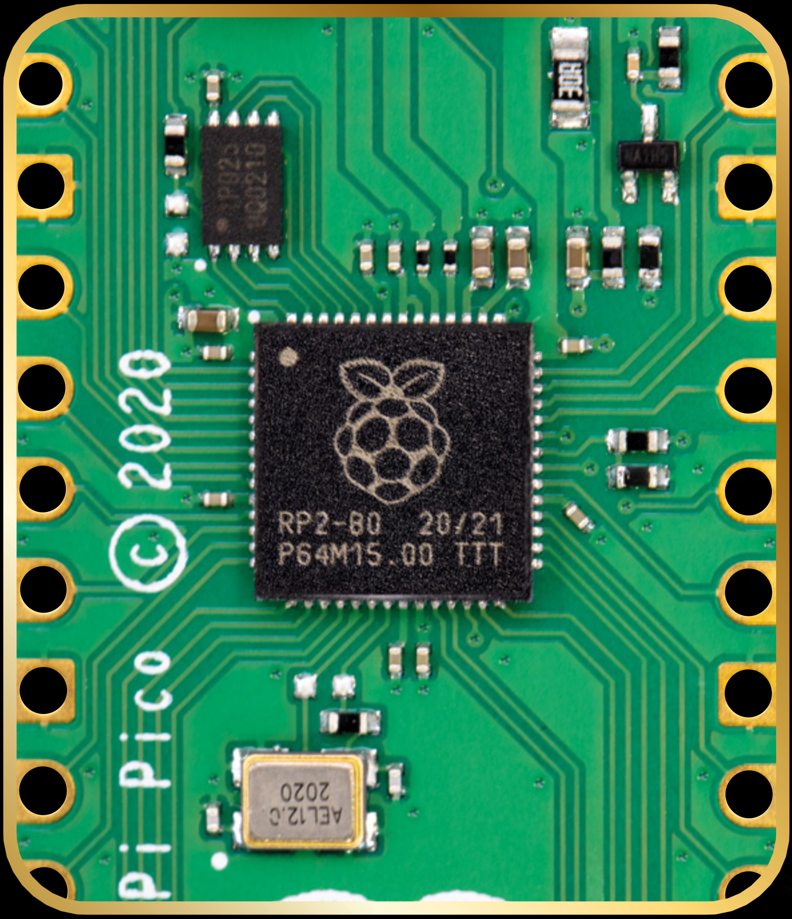
**James Adams, Pico design:** This chip is built on TSMC 40 nanometre process – the same process that the main processor on Raspberry Pis (except for Pi4) use. That's a relatively modern silicon process to use for a microcontroller. What that gives you is small transistors. It's been around since about 2008/2009 so it's quite mature, and it's not that expensive these days.

Because of the small transistors and the high speeds, you get very fast edges on your signal transitions. So the frequency content of those edges is high. So what the grounds are doing … if you have a signal pin going out to your circuit, you will also have a corresponding ground current coming back – it's just physics. The idea is that if you have a long loop where the current has to go out and come back in again, that is effectively an aerial. The other thing is that if you have multiple signal currents going out and a single ground pin, all of the signal currents are coming back through that single pin.

There are two things [these ground pads are] doing here. One is separating the return currents, which reduces cross-talk in signals. So if you've got a big old bus waggling about and only a single ground pin, a lot of those signals will get cross-talk because they're going through the same ground. This is not ideal from a circuit perspective. But also, they'll radiate more. Those are what will emit RF. If you want to build products that don't radiate, to pass your class B radiation compliance (that you have to do if you build real products), you don't want things radiating. That's a very long-winded way of saying that if you want fast signals, you need lots of grounds. You want to minimise your current loops, and you want to minimise your cross-talk.

Also, it's a useful thing to have. When attaching things to the GPIO you often need a ground pin too. It's a useful thing to have irrespective of all the engineering-y reasons.

# RASPBERRY PI PICO MICROCONTROLLER



> “ RP2040 IS MORE THAN CAPABLE FOR MOST USE-CASES ”

## MEET THE TEAM

**HS: What is the process of making a new microcontroller?**

**Nick Francis, Hardware design:** We knew we wanted to do a microcontroller, and we knew we wanted to do something small and cheap and fast. The M0+ is a tiny, tiny, tiny microcontroller … quite early on we decided that dual core was a simple thing to do for us. They’re so small!

Early on, we had discussions like: ‘How many UARTS? How many I2Cs do you want?’ Do we want two or three or six, or whatever? Then we started to throw more and more gates at it. Instead, we decided to build PIO [see overleaf for details] so – I did some early architecture work on that trying to build a completely flexible IO block. Luke [Wren] then took that to another level, and it’s become a much more flexible beast that can really be used to emulate any peripheral that we would want to use.

The PIO then became a bigger and bigger part of the chip, so we adjusted some of the architecture. We put in a much more flexible DMA controller that was super-efficient and highly tuned for targeting working with the PIO, and other peripherals. We were trying to use every gate we could – use every cycle to get the most performance out of the chip.

# RP2040 IN DEPTH

## FIND OUT WHAT GOES ON INSIDE THIS MICROCONTROLLER

### PROCESSING POWER

RP2040 has two 32-bit ARM Cortex-M0+ processors running at up to 133MHz (or even more if you want to try overclocking). Microcontroller performance is a bit more nuanced than, say, PC performance because size, power, and cost trade-offs are often important.

RP2040 is more than capable for most common microcontroller use-cases: you should find that it has plenty of horsepower. Additionally, RP2040 has optimised floating-point routines in the boot ROM which give it a speed boost.

### LOW POWER

The Cortex M0+ cores in Pico don't suck up a lot of power when running normally, but if you want your projects to run long-term on battery power, they may draw a little too much current. Fortunately, they have a couple of features to help get the most out of limited power sources. Sleep and Dormant modes can drop the power consumption of a Pico down to around 1 mA. In these states, the Pico can't do any processing, but it can wait for a particular event (such as time signal from the real-time clock, or a signal on a GPIO pin), then wake up and perform a particular task (such as read a sensor and save the value).

### DUAL CORE

There are two Cortex-M0+ processor cores in Pico, and each of them is capable of the same set of things. This, for example, allows you to dedicate one core to just running an interface (or set of outputs) while the other takes data in and processes it. Obviously, this lets you do more processing in a given time, but it can be just as useful for simplifying your code. If you've got something that needs constant attention (such as an animation you want to proceed smoothly, or a data stream that needs constant processing), you can dedicate one core to just this while the other core handles all the ancillary tasks. Your timing-critical code can then just run uninterrupted and the other bits can still get processor time.

### REAL-TIME COUNTER (RTC)

Pico includes a timer that can be used with an external reference clock to get the time in human-compatible units such as days, months, and years. This RTC can also be used to generate alarms at particular points. If you want, for example, a particular task to run every day at 10 am, the RTC can trigger it. Similarly, if you want something to trigger an action in a set amount of time (such as a 30-second timeout), the RTC can help.

### FULLY CONNECTED BUS

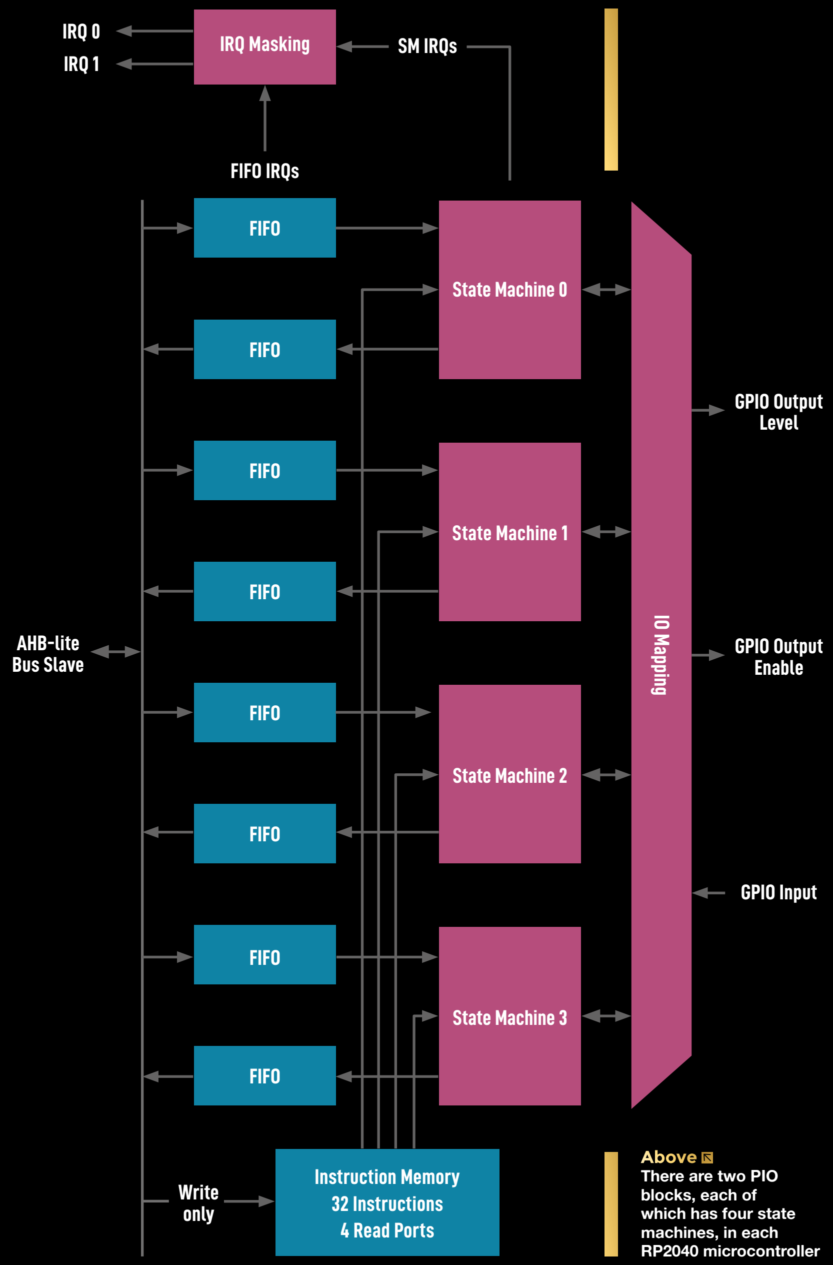
There's quite a lot going on inside the black package of RP2040 – including two processor cores, peripherals for many protocols, and PIO – and these all need to access memory. The fully connected bus fabric gives fast and predictable performance, even when lots of things are happening at once.

### COMMUNICATION PROTOCOLS

RP2040 is a great microcontroller, but you're likely to need additional parts to build up your projects, and this means you need a way of communicating with them. To make this quick and easy, RP2040 has hardware support for two I2Cs, two SPIs, and two UART buses. If this isn't enough – or you want to connect more esoteric hardware, you can use Programmable Input and Output (see next page). →

# PROGRAMMABLE I/O

## CONNECT PICO TO ALMOST ANYTHING


IRQ 0
IRQ 1
IRQ Masking
SM IRQs
FIFO IRQs
FIFO
State Machine 0
FIFO
FIFO
State Machine 1
FIFO
AHB-lite Bus Slave
FIFO
State Machine 2
FIFO
FIFO
State Machine 3
FIFO
IO Mapping
GPIO Output Level
GPIO Output Enable
GPIO Input
Write only
Instruction Memory
32 Instructions
4 Read Ports

**Above** There are two PIO blocks, each of which has four state machines, in each RP2040 microcontroller

**Like many microcontrollers, RP2040 has hardware support for some digital communications protocols such as I2C, SPI, and UART.** This means that if you want to communicate with something via I2C, you can simply send the raw data to the I2C peripheral and it will control the pins to handle the specific timing requirements for this protocol. However, what happens if you need something a little unusual such as WS2812B LEDs, or more I2C or SPI buses than are available?

On most microcontrollers, you have to 'bit-bang', which means using the main processing core to turn the pins on and off directly. This can work, but it can cause timing problems (especially if you use interrupts), and can take a lot of processing resources that you may need for other things. Pico has a little trick up its sleeve: Programmable I/O (PIO).

As well as the two main Cortex-M0+ processing cores, there are two PIO blocks that each have four state machines. These are really stripped-down processing cores that can be used to handle data coming in or out of the microcontroller, and offload some of the processing requirement for implementing communications protocols.

For each simple processor (called a state machine), there are two First-In-First-Out (FIFO) structures: one for data coming in and one for data going out. These are used to link the state machine to everything else. Essentially, they are just a type of memory where the data is read out in the same order it's written in – for this reason, they're often called queues because they work in the same way as a queue of people. The first piece of data in (or first person in the queue) is the first to get out the other end. When you need to send data via a PIO state machine, you push it to the FIFO, and when your state machine is ready for the next chunk of data, it pulls it. This way, your PIO state machine and program running on the main core don't have to be perfectly in-sync. The FIFOs are only four words (of 32 bits) long, but you can link these with direct memory access (DMA) to send larger amounts of data to the PIO state machine without needing to constantly write it from your main program. The FIFOs link to the PIO state machine via the input shift register and the output shift register.

There are also two scratch registers called X and Y. These are for storing temporary data.

The processor cores are simple state machines that have nine instructions:

- IN shifts 1–32 bits at a time into the input shift register from somewhere (such as a set of pins or a scratch register)
- OUT shifts 1–32 bits from the output shift register to somewhere
- PUSH sends data to the RX FIFO
- PULL gets data from the TX FIFO
- MOV moves data from a source to destination
- IRQ sets or clears the interrupt flag
- SET writes data to destination
- WAIT pauses until a particular action happens
- JMP moves to a different point in the code

> “ THERE ARE TWO PIO BLOCKS THAT EACH HAVE FOUR STATE MACHINES ”

Within these there are options to change the behaviour and locations of the data. Take a look at the data sheet for a full overview.

In addition, there are a couple of features to make life a little easier:

- Side-setting is where you can set one or more pins at the same time that another instruction runs
- Delays can be added to any instruction (the number of clock cycle delays in square brackets)

Let's have a look at a really simple example, a square wave:

```
.program squarewave_wrap

.wrap_target
        set pins, 1 [1]
        set pins, 0 [1]
.wrap
```

This uses an outer wrap loop. The **`wrap_target`** label is a special case because we don't need to do an explicit jump to it: we can use **`.wrap`** to jump straight to **`.wrap_target`**. Unlike an explicit jump instruction, this doesn't take a clock cycle, so in this case we're producing a perfect 50% square wave. The .wrap and .wrap_target labels can be omitted. Each instruction also has a delay of a single clock cycle.

The value of **`pins`** is set when we create the PIO program, so this can create a square wave on any I/O.

Let's take a look at a more useful example, an SPI transmitter. This just controls the data-sending line for SPI:

```
.program spi_tx_fast
.side_set 1

loop:
        out pins, 1 side 0
        jmp loop side 1
```

This shows the power of the side-set. In this case, the side-set pin is the SCL (clock) line of SPI, and the out pin is the SDA. The **`out`** instruction takes a value from the FIFO and sets the pin to that value (either 1 or 0). The next instruction triggers the side-set on the clock pin, which causes the receiver to read the data line. Notice that the second instruction is a jump, not the wrap as in the previous example, so it does take a clock cycle to complete.

The **`out`** instruction gets a bit of data from the output shift register, and this program would quickly empty this register without a **`pull`** instruction to get data into it. However, the **`pull`** instruction would take one cycle which would complicate the timings. Fortunately, we can enable autopull when we create a PIO program, and this will continuously refill the output shift register as needed as long as data is available in the FIFO. Should the FIFO be empty, the PIO state machine will stall and wait for more data.

The speed of this SPI transmit is set up the clock speed of the PIO state machine, and this is configurable when you start the PIO.

This has been a whistle-stop tour to show off the ideas behind PIO. For more details about exactly how to implement a PIO program, take a look at the tutorial on page 90 →

# OTHER RP2040 HARDWARE

## OTHER WAYS TO USE THE MICROCONTROLLER

**Pico isn't the only board built on an RP2040.** The microcontroller chip is available for other companies to build products off, and here are some other boards that are coming soon. They're all based on the same underlying processing power, but they each build on it in different ways to make products for different uses.

## PICO SYSTEM

**PIMORONI | £58.50 | pimoroni.com**

Unlike the other boards we've looked at here, Pico System isn't intended to be a general-purpose board; it's designed with one (very obvious) use in mind: RP2040 makes a great gaming platform. PIO with DMA means that it can output video data very quickly with little CPU overhead, and having two cores and plenty of memory means that there's loads of scope for doing all the processing that games demand. Add to this some beautiful design work, and you get a fantastic handheld console which we're looking forward to playing with more in the coming months.

PICO SYSTEM
Pocket-size 32-bit dual-core gaming system with 240x240 IPS screen etc.
Pimoroni PicoSystem
pimoroni.com
Made in Sheffield, UK
海賊ロボ忍者さる

**Left** This is a prototype Pico System, so the final version may look a little different

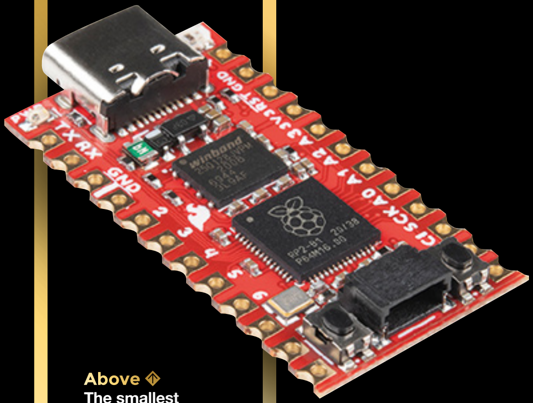
> “IT'S DESIGNED WITH ONE (VERY OBVIOUS) USE IN MIND”

## PRO MICRO RP2040

**SPARKFUN** | $9.95 | **sparkfun.com/rp2040**

If you're looking for the smallest RP2040 board, then this is it. SparkFun has squeezed and shrunk everything down to the footprint of its Qwiic Pro Micro range. However, unlike the original (which features a 16MHz, 8-bit microcontroller), this has all the processing power of the RP2040 to get your project running. Alongside this, there's 16MB of RAM for your programs and data, which should be enough for almost all microcontroller applications.

Despite its small size, it still breaks out 18 of the GPIOs, including four analogue inputs, and has a Qwiic connector for additional hardware. There's a USB for data and power, a power LED, and a user-controlled RGB LED. There are reset and boot buttons, so you can program one of these without unplugging power (as is needed on Pico).



**Above** ◈
The smallest RP2040-based board available at the moment

> “ DESPITE ITS SMALL SIZE, IT STILL BREAKS OUT 18 OF THE GPIOS ”

## MICROMOD RP2040 PROCESSOR BOARD

**SPARKFUN** | $11.95 | **sparkfun.com/rp2040**

We looked at the MicroMod system in issue 38. It allows you to quickly and easily slot a microcontroller module into a carrier board. These modules are stripped right back to just the bare essentials, so they're tiny, and the M.2 mounting means they are secure and don't add any height to the carrier board.

This can mean you have a bit less flexibility than with some other add-on systems, but if there's a carrier board that suits your needs, it can be a compact and robust solution. If you're looking for a large Arduino Uno-like board with an RP2040 at its heart, there's the ATP carrier. If you're looking for a board with input and display, there's the aptly named Input and Display carrier. There are also data logging and machine learning carrier boards.

As with the Pro Micro, there's also 16MB of flash memory to store programs and data, but other than this, all features are on the carrier boards rather than the processor boards. →

**Below** ◈
Slot an RP2040 into the heart of other carrier boards

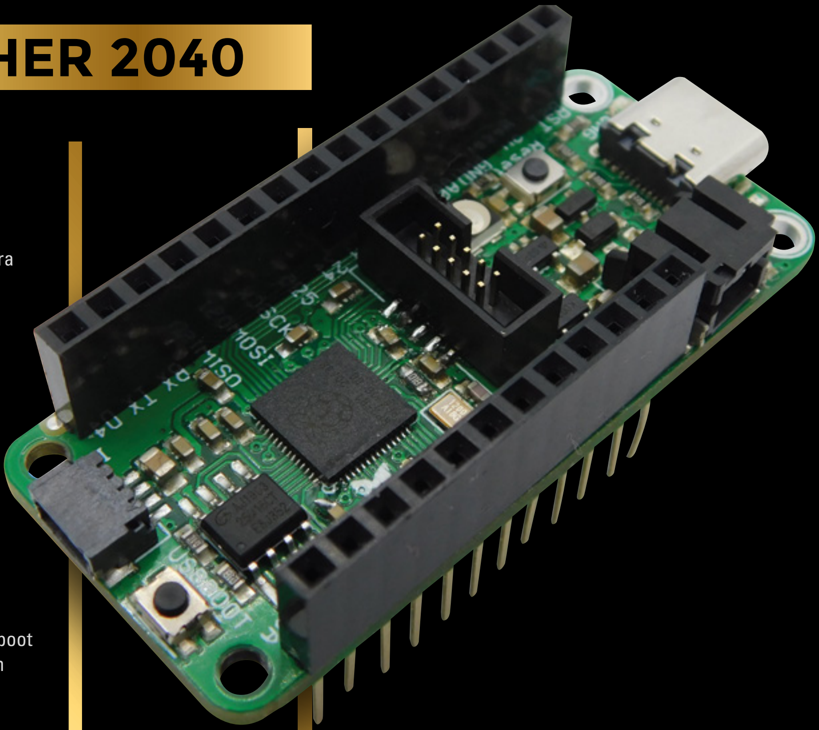
## ADAFRUIT FEATHER 2040

**ADAFRUIT** | $9.95 | **adafruit.com**

You're almost certainly going to want to plug some extra hardware into your microcontroller to build your projects. Whether that's sensors, displays, or actuators, it's this extra hardware that is usually the point of microcontroller projects. There are already some great accessories for Pico (see next page), but if you want to get access to a huge range of add-ons, the best solution is to jump right into one of the biggest microcontroller ecosystems around – Feather. Adafruit's Feather 2040 has the standard pinout that lets you plug in almost any Feather add-on (or Wing as they are known). These are stackable, so you can add several of them if you need lots of features for your project.

21 GPIOs are broken out, and there's also on-board LiPo battery charging if you need your project to be portable, and a STEMMA QT connector for adding even more hardware. There's 4MB of flash for storing your data and programs, and like the Mini, there are both reset and boot buttons. All this means that the board is slightly wider than Pico (one row on a breadboard).

The Feather 2040 comes set up and ready to run CircuitPython. This is a little different to MicroPython on Pico. CircuitPython includes excellent support for Adafruit (and other) hardware.



**Above** Access a huge range of hardware add-ons through the Feather ecosystem

## THING PLUS RP2040

**SPARKFUN** | $17.95 | **sparkfun.com/rp2040**

SparkFun's Thing Plus is also compatible with Feather, so you get access to that ecosystem of add-ons. However, the Thing Plus is slightly longer than the classic Feather. There are 19 exposed GPIOs (two fewer than on some Feather-compatible boards), but there is a Qwiic connector (next to the battery charger) to give even more options for connecting hardware, and a microSD card holder (on the underside) for storage. Couple this with 16MB of flash and you've got enough storage for almost any eventuality. For a full range of blinkies, there's an LED on pin 25 (the same as on Pico), and a WS2812B LED to give full RGB colours.

**Right** It's a small detail, but the reset and boot buttons on the Thing Plus feel great to use

> "THING PLUS IS ALSO COMPATIBLE WITH FEATHER"

# PICO ADD-ONS

## PLUG NEW FEATURES INTO YOUR PICO

**You can add hardware to your Pico by building your own circuits – and in many cases, this is relatively straightforward thanks to protocols such as SPI and I2C.** However, sometimes it's easier and quicker to start with a premade add-on that gives more capabilities to your Pico so that you can spend your time working on other parts of the project. We expect there to be lots of add-ons for Pico created in the coming months, but here are some that you can get right now from Pimoroni.

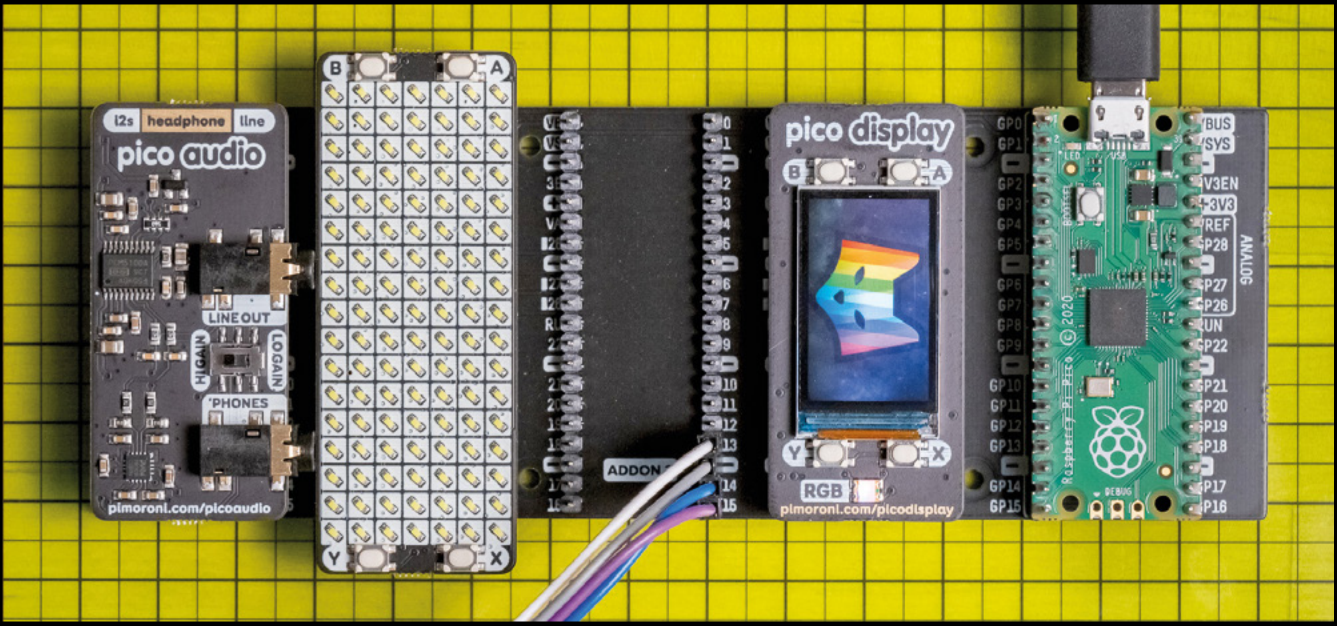
Pico Audio adds an I2S decoder and headphone and line-level outputs to Pico. Pico has a capable audio system that can output I2S via PIO. You can create buffers of data that are streamed to your attached device. At the time of writing, this is in the pico-extras repository (**hsmag.cc/pico-extras**) because, while it does work, there is ongoing development and the API may change. You can use the audio system to directly output audio using PWM, but you can get better quality using I2S, and the integrated headphone and line-level amps make this easy to use.

If you prefer visual output rather than audio, Pico Display gives a TFT screen, four buttons, and an RGB LED, all within the Pico footprint. The result is a compact, encapsulated display, perfect for adding a little visual output to your project.

For a bit more sparkle, Pico Scroll adds a 17×7 matrix of white LEDs to the top of the Pico. As the name suggests, this works well for scrolling text as well as displaying data or just generally shimmering. There are four buttons to get data in from the user to control the display, or perhaps play a game of Snake.

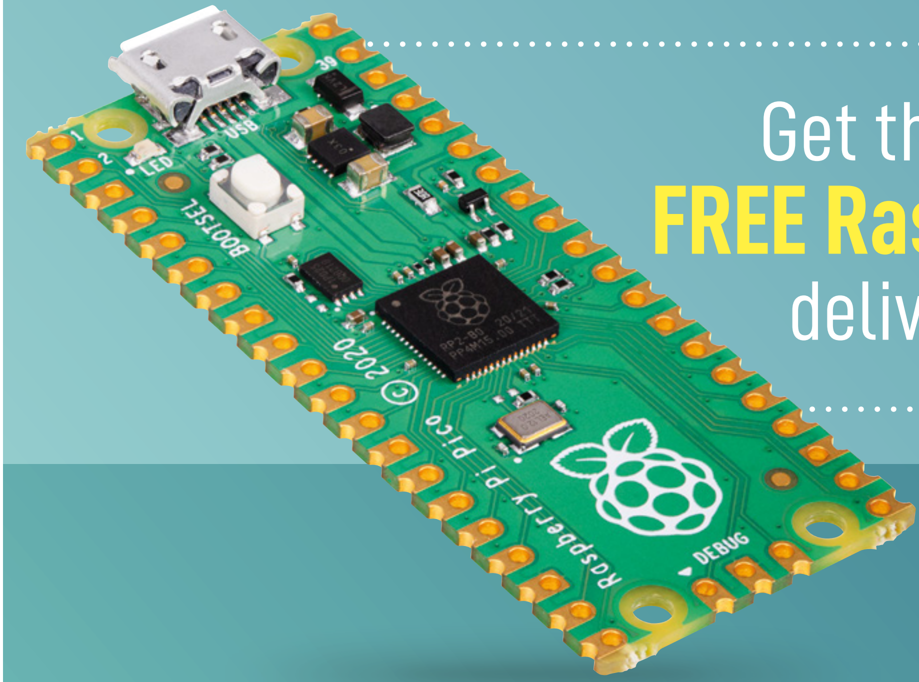
Of course, you may want more than one of these, and for that you'll need an Omnibus (for two add-ons) or a Decker (for four add-ons). You can also use these to access all the pins when another add-on is on top, whether that's for debugging or expanding your project. ◻

**Below** Pico Audio, Scroll, and Display add-ons from Pimoroni, linked together on a Pico Decker


i2s
headphone
line
pico audio
LINE OUT
HI GAIN
LO GAIN
PHONES
pimoroni.com/picoaudio
B
A
Y
X
ADDON
pico display
RGB
pimoroni.com/picodisplay
Raspberry Pi Pico
© 2020
DEBUG
VBUS
VSYS
3V3EN
3V3
VREF
GP28
ANALOG
GP27
GP26
RUN
GP22
GP21
GP20
GP19
GP18
GP17
GP16
GP0
GP1
GP2
GP3
GP4
GP5
GP6
GP7
GP8
GP9
GP10
GP11
GP12
GP13
GP14
GP15

# SUBSCRIBE TODAY

## FOR JUST £10



USB
LED
BOOTSEL
2020
Raspberry Pi Pico
DEBUG

Get three issues plus a **FREE Raspberry Pi Pico** delivered to your door

hsmag.cc/FreePico

UK offer only. Not in the UK?
Save money and get your
issue delivered straight to your
door at **hsmag.cc/subscribe**.
See page 66 for details.

Subscription will continue quarterly unless cancelled

# SUBSCRIBE on app stores

From £2.29

Available on the App Store

GET IT ON Google Play

**Buy now:** hsmag.cc/subscribe

Free Pico with print subscription only

FREE DUAL-CORE MICROCONTROLLER
HackSpace
TECHNOLOGY IN YOUR HANDS
INTRODUCING
RASPBERRY PI
PICO

SAVE 44%

FREE DUAL-CORE MICROCONTROLLER
HackSpace
TECHNOLOGY IN YOUR HANDS
hsmag.cc | February 2021 | Issue #39
INTRODUCING
RASPBERRY PI
PICO
LOW-COST · HIGH-PERFORMANCE · FLEXIBLE I/O
A NEW MICROCONTROLLER BOARD FROM RASPBERRY PI
MAKE | BUILD | HACK | CREATE
PLUG AND PLAY
Billie Ruben
DOOR BELLS KEYS FACE R
RASPBERRY PI 400
3D DESIGN
SELECT THE PERFECT MOTORS
PICK YOUR CHASSIS
OMATION 3D PRINTING MUSIC

# How I Made
# AN AUDIO DECODER

Turning light into sound

**By Emily Velasco**

**As modern humans, we not only stand atop the shoulders of the giants who came before us, we walk amongst their bones.** The detritus of their past lives and technologies long since rendered obsolete swirl around our feet, forgotten, overlooked, and often discarded.

But sometimes the voices of our ancestors linger behind, trapped within scraps of the world that they left. And sometimes, if we're willing to listen closely, we can hear what they're trying to say.

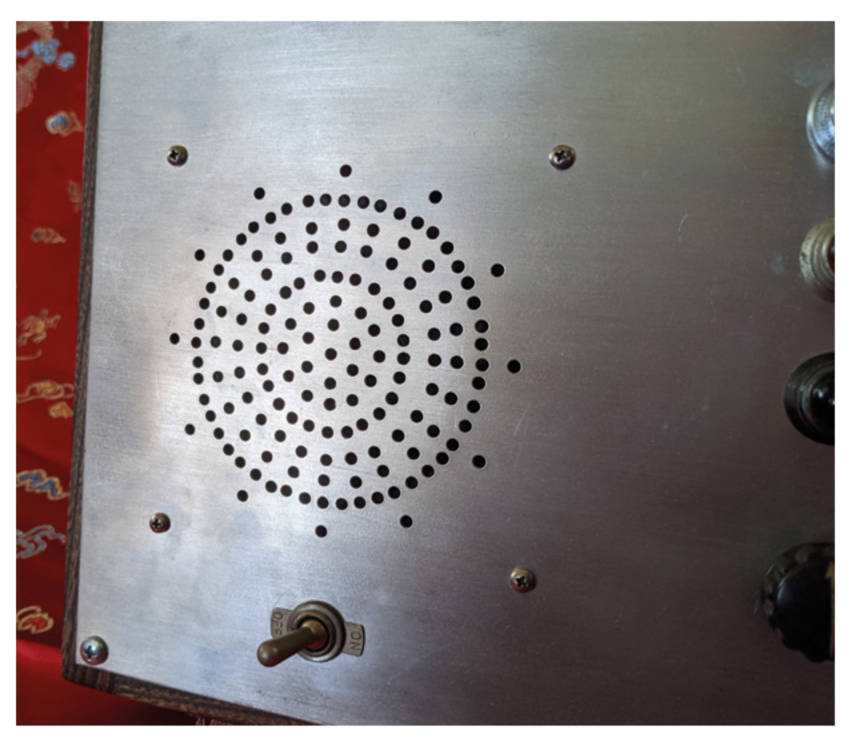
## STAYING SANE IN QUARANTIMES

For a moment, let's rewind to 2019. Back in the days when office workers worked in offices, one of my co-workers, knowing my propensity for collecting old and unusual things, asked me if I wanted a 16 mm film projector that had belonged to his father. I, being presented with the opportunity to indulge in my favourite vice of hoarding, enthusiastically said yes. It turned out to be a gorgeous, albeit only partially functioning, bit of 1930s technology. I spent some time repairing it, and then I sought out a film to play on it. A trip to the local flea market bore fruit a few months later, and I came home with a mid-century educational film on man-made isotopes.

Looking at that film, I noticed something I hadn't expected. Running down one of its sides was a track with a curiously squiggly line. It was the audio for the film, stored in optical form.

I had never really pondered how old films captured their audio, but of course, it would be easiest to capture sound as light. The rest of the film held its information optically, so why not encode the audio in the same medium?

My projector, however, came from a time when films were still likely to be of the silent variety, and it had no way of playing back the audio on this flea-market film. I briefly toyed with the idea of adding



**Left**
**My radiopunk speaker grill**

some electronics to the projector to play that audio, but that notion was quickly swept aside by newer and shinier project ideas and soon forgotten.

A year later though, I was scraping the bottom of my project barrel in search of something to do to keep me sane through another day, week, or month of quarantining, and that faint ember of an idea reignited.

'How hard could it be to build something to decode that optical sound track?' I thought to myself.

Heh, famous last words.

## THE OLDEN DAYS

Before I get into my thought process on building an optical audio decoder, I should probably explain how optical sound tracks work.

The squiggle running down the centre of the track is a visual representation of the sound that goes with the film. Every one of its undulations corresponds to a sound-wave that wiggled its way through the air at the time the film was recorded.

To turn those undulations back into something we can hear, movie studio engineers of a hundred years ago put a light source on one side of the film, and a light receiver on the other side of the film. As the sound track slides between them, some of the light shines through the squiggle and lands on the receiver on the other side. Since the squiggle varies in width, the amount of light landing on the receiver varies in a way that corresponds to the sound that was recorded. With the receiver translating this light signal into an electrical signal, one only needs an amplifier and a speaker to further transform it into something audible.

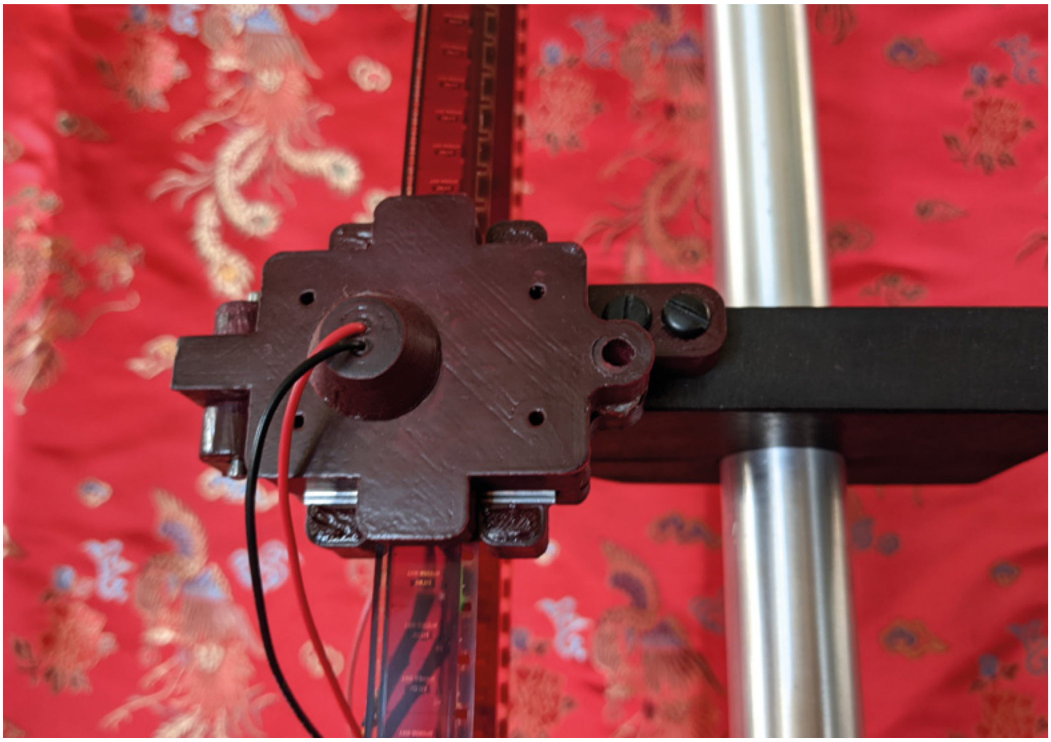
## DECODER DESIGNING

When that technology was first devised in the late 1920s, the engineers did all of this with incandescent bulbs and specialised vacuum tubes, but I, a home tinkerer a century later, had more options.

My thought was that I could put an LED on one side of the film, and a photoresistor on the other side. I have literally hundreds of each at my disposal. Easy, right? In theory, yes. In practice, well… →

FWD
STOP
REV

**Right**
**Letter punches let you add text to a surface**

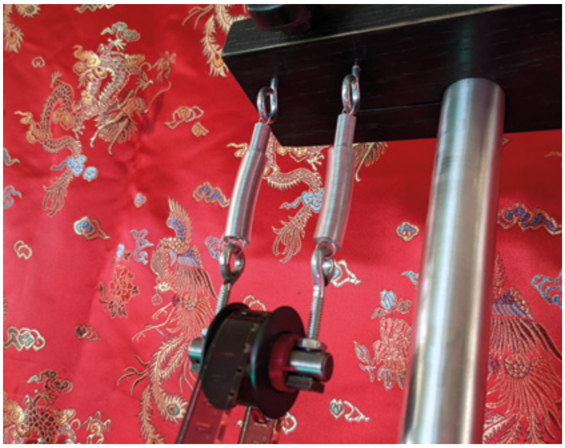


**Left**
**A 3D-printed part holds everything in place**

Let's go back to the squiggle. If we look closely at it with the aid of a magnifying glass, we will see that within its many undulations, there are even tinier waves, many thinner than the breadth of a human hair. If we simply cast light as wide as an LED on that, our receiver will translate the sum of those waves into sound all at once. The audio will end up indistinct, in much the same way it's difficult to understand what a friend is telling you while hanging out in a crowded room with dozens of conversations happening.

To hear the audio clearly, we need to shine light on just one tiny undulation at a time. I'll admit that coming up with a way to do this baffled me a bit. I had an idea churning around in my head about trying to cut a thin slit in aluminium foil when a friend asked me if I had considered using disposable razor blades. Razor blades, he said, are cheap, have a precisely honed edge, and are used in some optics experiments to create a narrow slit for laser light to pass through. Perfect.

Working in CAD software, I designed some parts to hold an LED, a photoresistor, the film, and the razor blades, and I fired up the 3D printer. Hours later, I had a prototype. I installed the electronics, added some soft felt to be gentle on the film, clamped it in my vice, inserted a manageable length of film, attached a battery, pulled the film through and … nothing. I didn't get a single sound. OK, back to the drawing board.



**Left**
**Tensioning springs hold the film in place**

My research suggested that maybe a photoresistor was not sensitive enough to work with the minute amount of light that passes through the narrow slit between the razor blades, so I went digging in my parts bins for a photodiode. Photodiodes are more sensitive than photoresistors, and they also respond more quickly to changes in frequency, so I figured maybe that would work better than the photoresistor had. I jumped back into the CAD software and redesigned my parts to hold the photodiode instead. Hours later, I had a second prototype. I assembled everything a second time, and headed out to the workshop with version two.

I clamped it all in my vice again, attached the battery, inserted the film, and … nothing. I was starting to get a bit discouraged, but on a whim, I attached a set of earbuds to the amplifier, stuffed

**Left**
**Panel mount LEDs make great retro-looking indicators**

**Above**
**This can play the audio backwards because...why not?**

them in my ears, pulled the film through the device, and I heard something! It was faint, but it was there. In my excitement, and to make sure I hadn't imagined it, I pulled the film back and forth through the device several more times. It was muffled and indistinct, but unmistakable. I had my first hints of sound coming back to life from the decades-old film.

I tweaked some things, narrowing the gap between the razors even more, swapped in a brighter LED, and pulled the film through again. Goosebumps rose on my arms. The sound was still extremely quiet, but I could just begin to make out something that sounded like a voice.

## LEAVING FAMILIAR TERRITORY

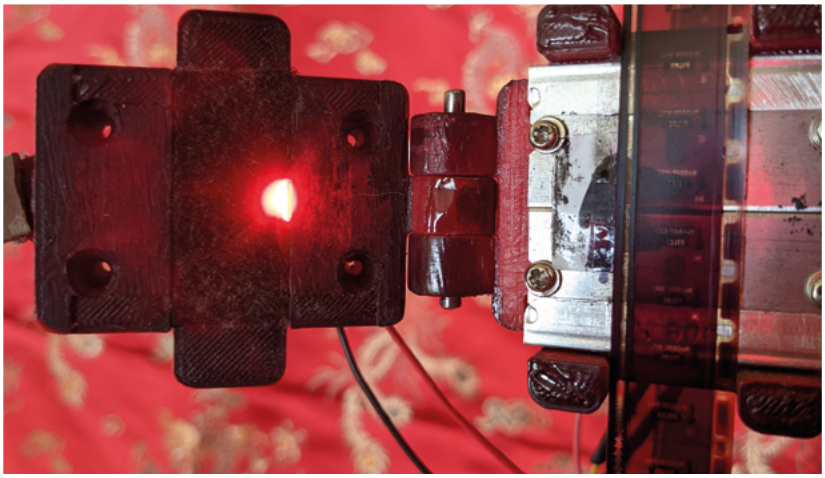
It was a promising start, but still a long ways from anything remotely intelligible. I needed to do more thinking. One night a few days later, I was chatting with a friend about the project, and he offered up something known as a transimpedance amplifier as a suggestion. He said it would be more sensitive than my photodiode, and tried explaining how they work and how to build one but I, not being an electrical engineer, might as well have been a brick wall.

'Don't worry,' he said. 'You can just buy one. Go look for an OPT101.'

I went looking and found out a couple of things. For one, I learned that an OPT101 is an integrated circuit that combines a photodiode and an op-amp in one package. For two, I learned I could get them for a few dollars apiece online. I ordered a couple and waited.

They arrived about a week later, and I got back to work. I redesigned my parts in CAD, printed them, installed the electronics and razor blades, and headed back out to the garage. With the contraption held in my vice yet again, I pulled the filmstrip through and out of the speaker came a loud, buzzy voice. This time I could almost make out what it was saying – something about sodium. Goosebumps again.

I spent some more time making adjustments and tweaking things, and pulled the film again in what was becoming almost a ritual. →



**Right**
**An LED supplies light in place of the traditional bulb**

From the speaker came the voice of a man with a British accent, and he said:

"If a compound containing sodium is inserted into the reactor..."

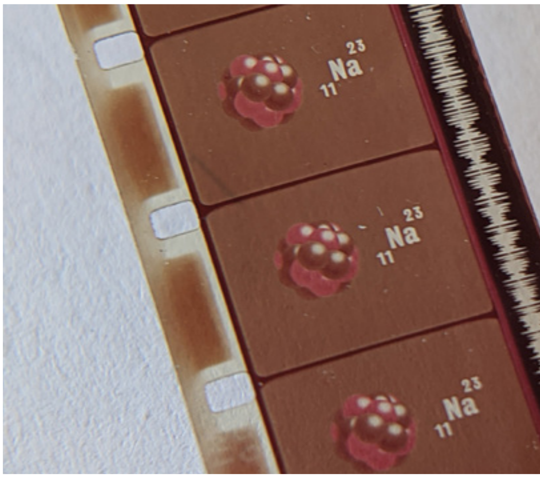
I couldn't help but laugh with excitement. I almost couldn't believe it. I had actually done it. This man, who was probably decades in the ground by now, was speaking to me in my very own garage. Someone once called me a techno-witch, and for once, I felt like it was true. It was like I had performed a séance, and here was this ancient disembodied voice explaining to me how to make sodium isotopes.

## ARTISTIC LICENCE

When I began the project, I hadn't really considered where it would go beyond this point. I just wanted to see if I could decode an optical sound track, and I had. It was cool as hell though, and I figured I should do something with it. I spent the next few weeks indulging my creative impulses, and bit by bit, things came together. I didn't want to deal with dozens or hundreds of feet of film, so I decided to make loops of film similar to how some electronic

**Left** A wooden box houses the audio parts of this project

**Below** The audio is the wiggle on the right-hand side

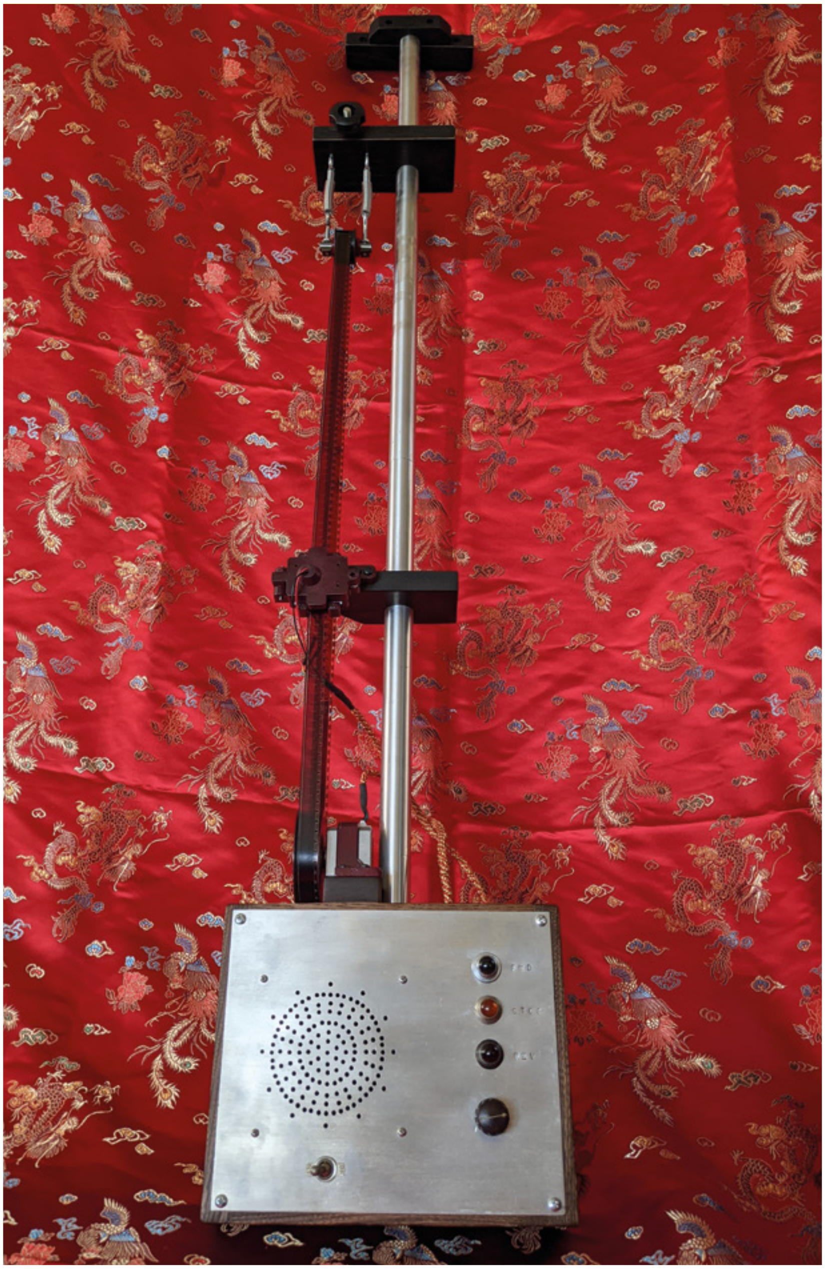


musicians use loops of tape from an audio cassette. Because the technology for playing optical sound tracks originally dated to the late 1920s, I wanted the project to have the look of what, for lack of a better term, I'm calling radiopunk – that aesthetic of darkly stained wood, chunky bakelite knobs, and polished metal you find on very old radio sets.

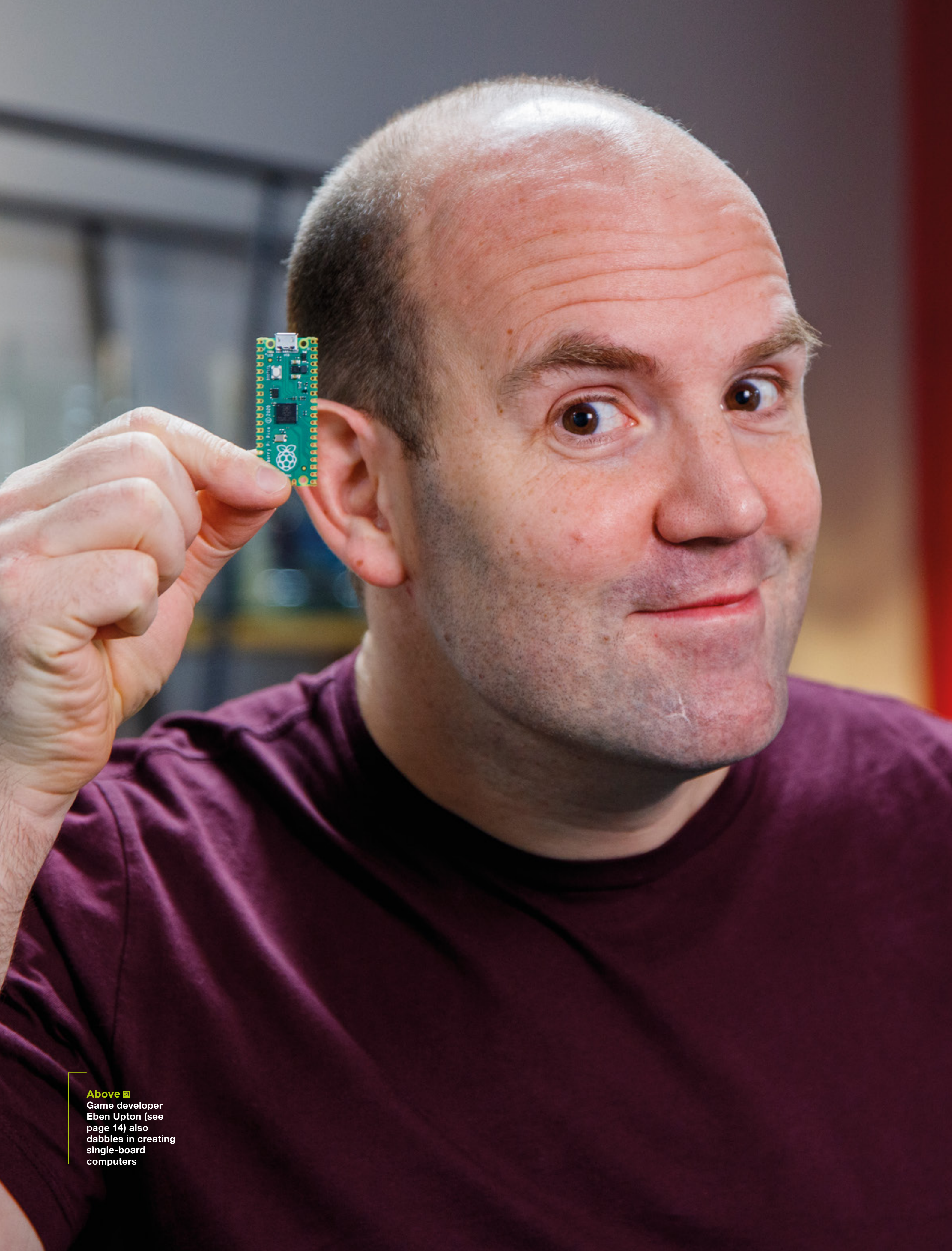
To drive the film, I chose to use a stepper motor because it's a simple matter to control their speed in a precise way using an Arduino, which is one microcontroller I can actually write code for. This was the easy route, but it was maybe not the most ideal solution either. Stepper motors, as their name suggests, move in steps. Those steps mean the motor's motion isn't perfectly smooth, and in the decoder, it means there is some audible distortion when it plays a sound track. I could probably solve that with more sophisticated electronics, or through some additional mechanical parts, but sometimes you just have to call a project done. I'll save those ideas for my next optical audio project, and if anyone asks about the motor noise on this project, I'll tell them it's part of the aesthetic. That's one advantage of people thinking you're an artist. You can make things up, and they have to believe you.

Now that it's done, I don't really know what I'm going to do with it. I suppose I'll hang it on the wall in my living room, and if guests come over, they can try it out, and learn how to make isotopes, one film loop at a time. ◻

> I had actually done it. This man, who was probably decades in the ground by now, was speaking to me in my very own garage



**Right** The finished project ready to be wall-mounted



**Above** ↗
Game developer Eben Upton (see page 14) also dabbles in creating single-board computers

HackSpace magazine meets...

# Eben Upton

The boss of Raspberry Pi explains what this Pico malarkey is all about

**After years of blood, sweat, and tears, we can finally talk about Raspberry Pi Pico in public.** It's the first entirely new product from Raspberry Pi, designed entirely in-house to do a specific job: put the ability to make smart devices into as many hands as possible. If you want the technical details, page 30 is the place for you. If you want the philosophical take on things, who better to talk to than Eben Upton, Raspberry Pi's benevolent overlord? We spoke to him from his hollowed-out volcano lair deep in the Fens about what Pico is, what it does, who it's for, and why it was worth spending millions to develop it. →

**HackSpace** **First of all, what is Pico? Is it just the latest version of the Raspberry Pi?**

**Eben Upton** It's a microcontroller board. The easiest way of differentiating it from a Raspberry Pi is that, unlike a Raspberry Pi, it's not a platform you develop on; it's a platform you develop for. So it needs a host, and Raspberry Pi is a good host.

Everything we make is about bringing more capabilities to people who wouldn't have thought of themselves as engineers. Raspberry Pi has been very successful at that, because it allows people to go and do embedded stuff. The whole world of [Cortex-] A class devices – Raspberry Pi and its maker competitors – does that really well. Pico is about bringing the same capabilities to people who want to get to an even lower level. So people who want to run code on bare metal...

Actually, you can run code bare metal on Raspberry Pi, but it's designed to have a big operating system in the way. Which is good, because it brings lots of features, but is bad because it brings, for example, a lack of deterministic latency. That's the nice thing about microcontrollers. You run a program and it sits there polling a GPIO and the moment the GPIO changes, it goes and does something else. When we say 'the moment', you can localise what 'the moment' is down to a handful of nanoseconds. Which you can never do on a proper computer, right?

So it brings deterministic latency, analogue input, and Pico can go down to very low levels of power consumption.

Also, it's mechanically smaller than everything else we make. Even the board itself is mechanically smaller than everything else we make.

**HS** **How does it fit alongside the rest of Raspberry Pi's wares?**

**EU** It's useful to put it alongside Raspberry Pi 400 and Compute Module 4 actually, in terms of what it does to the business. Each of them is an extension of Raspberry Pi in the direction that we're already travelling in.

Compute Module 4 is about people who want to do the high-performance embedded stuff that Raspberry Pi has become very popular for, but in their own form factor with more customisation that you can't do in a core product.

Raspberry Pi 400 is about using Raspberry Pi as a PC. It's for those who want a uncompromised PC experience.

Pico is going in the other direction: people who want to be even more deeply embedded than a Raspberry Pi or a Compute Module – what can we give to them?

> "It's ideal for jobs like controlling strings of LEDs, polling sensors, and writing values on to SD cards"

**HS** **That sounds like it's aimed at people who are already comfortable with creating embedded devices?**

**EU** It's for new developers as well as more experienced ones: that's why you can choose to use it with either C or MicroPython.

We are not recommending that your average Joe's first exposure to this product is using C. Those days are just behind us now. The C side of things is for people who are already familiar with embedded devices; the MicroPython side is for beginners.

We've split the market into two bits: there's a bit we're addressing with C, and for that, we want the best possible C SDK, not a port of somebody else's. And then you've got beginners, and the right answer for beginners is a high-level language, particularly MicroPython. It's a conscious choice, and it's one that we made to ensure that access to Pico is as easy as possible.

**HS** **What sort of things do you envisage people will be doing with their Pico after they've had a play with it?**

**EU** I hope they'll be using them for... all the things they do with other microcontrollers. There's that whole sea of things that are a bit low-power and a bit small. It's ideal for jobs like controlling strings of LEDs, polling sensors, and writing values on to SD cards. I genuinely think there'll be a subset of the things people do with Raspberry Pis, and people will migrate from Raspberry Pi to Pico.

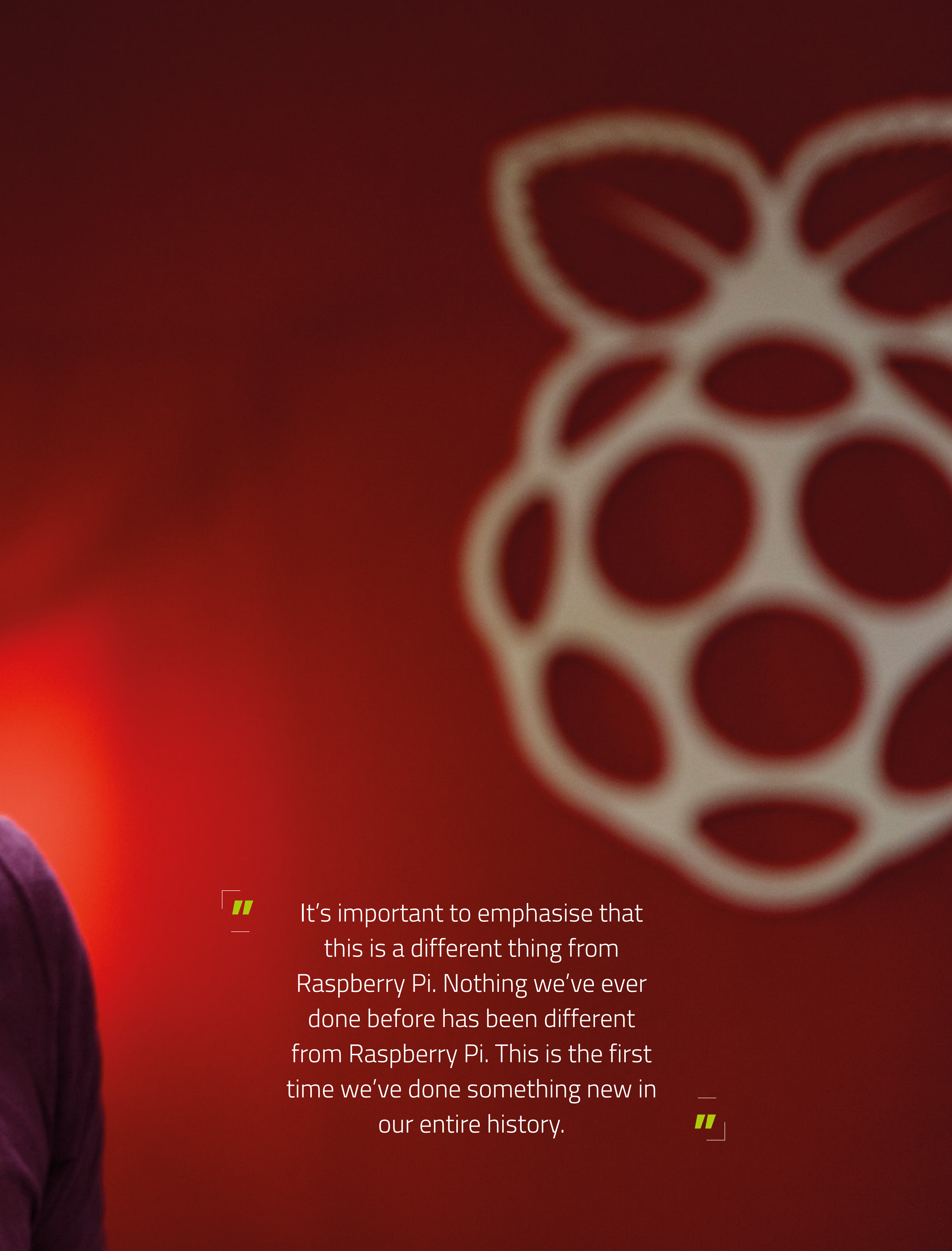
If you want to make a logging weather station, for example, maybe you want to do it on a Pico, because it'll consume less power. You can program it in Python, connect all the same sensors to it, and in fact, there'll be some analogue sensors that you couldn't connect to it using a Raspberry Pi. All these sorts of maker projects are ideal for a Pico – the maker projects that care more about power and size than about performance.

**HS** **And presumably the projects that don't actually need a full computer operating system?**

**EU** There's lots of stuff. If you want your environment for doing computer vision or something, probably the Raspberry Pi is the platform for that, because you can install OpenCV and a bunch of other stuff really easily and start working away really quickly, far more quickly than you could on a Pico.

But on the other hand, you can run off a battery for a lot longer on a Pico. So sometimes there'll be a project that will run just as well either on a Pico or on a Raspberry Pi, then it's a straight trade →

**Above**
“The right [language] for beginners is a high-level language, particularly MicroPython”



> It’s important to emphasise that this is a different thing from Raspberry Pi. Nothing we’ve ever done before has been different from Raspberry Pi. This is the first time we’ve done something new in our entire history.

between the size of the battery and the number of hours of programming that it takes to do it. The Raspberry Pi is really easy to hack stuff together on, but at the end of the day, you can't get away from the fact that you're running it on a PC.

**HS Do you have a comparison for how much longer a device will last if it's built on a Pico compared with, say, a Raspberry Pi Zero?**

**EU** I think that 10× is plausible. You're talking fractions of milliwatts rather than fractions of a watt. You can beat a Zero down to a few hundred milliwatts, but you can beat this thing down to a few milliwatts. I'd say 10–100× longer, depending on what you're doing.

If you run a Zero flat out, it will consume a watt and a bit, it won't be two watts. Probably not less than a watt. If you overclock a Pico and you run it flat out, then you could consume 100 to 200 milliwatts. So I guess the answer is that it depends on how much time spent idling the device is going to be. If they're running flat out, there's a factor of 10 between them, whereas if they're both idling, the Pico will consume roughly 100 times less power.

**HS What are the key features of the Pico?**

**EU** First of all, there's the price. Cost is about accessibility. You don't want a world where either people have to pay $20 for a microcontroller board, or else they can't get one, or that they only get one, or a world where they're then forced to buy clones of products that are of dubious quality. We want to provide quality and cost together in one place.

We have 256kB of RAM, 2MB of flash, and two cores at 133MHz. But the big feature that it's easy to overlook is the price. In that respect, I think you can see it being competitive with unbranded microcontroller boards. What was a really key goal with this product was to be cheaper than anything you can buy on AliExpress. Because it's about access.

We're bringing the Raspberry Pi brand values to the microcontroller board space. And what are our brand values? Well, we make very high-performance products. We make very low-cost products. And we make really well-engineered products. It's always those three with Raspberry Pi products. You have to be cheapest, you have to be the best, and best is fast and beautifully engineered. And Pico is beautiful, because it's another James Adams project.

The way the busing works inside the system, you can have one Arm core writing memory flat out, and one Arm core running memory flat out, you can have the DMA engine copying one bit of memory to another, and that can all happen at the same time.

Everything can happen at the same time because of the way that we have a fully connected switch. There are no bottlenecks in the chip, so it can run at full tilt.

What else? Having two completely symmetrical processors. There are a lot of multi-core processors, but they tend to be asymmetrical. They tend to have an application core, an M4 maybe, and a peripheral management core, an M0 maybe. We've gone for a different approach, where everything about the architecture is smooth and symmetrical. And very easy to understand. Because of the way the arbitration works, you can reason very precisely about what this chip is going to do on any given clock cycle.

**HS Why is that important?**

**EU** Well we've got performance, quite a lot of MIPS and memory, but we've also given you predictability. You can go close to the edge without falling off. If you have some real-time task, you can go right up and use 100% of the performance.

If you have non-determinism, you end up programming in a margin to account for that. You can't run the system flat out, so at say 70% of performance it starts to slow down, you know why and you know what you have to do. This one's designed to perform right to the limits of its capacity.

**HS Why release this now?**

**EU** It's important to emphasise that this is a different thing from Raspberry Pi. Nothing we've ever done before has been different from Raspberry Pi. This is the first time we've done something new in our entire history. The first time we've done something completely new.

Almost everything else we do is about making Raspberry Pi better, making accessories for it, polishing the platform. And while it's satisfying, it's also very linear, very incremental.

So, after nine years polishing one platform, it just felt like time to do something different. It's been a complicated programme, an expensive programme, but I think it's been worth it.

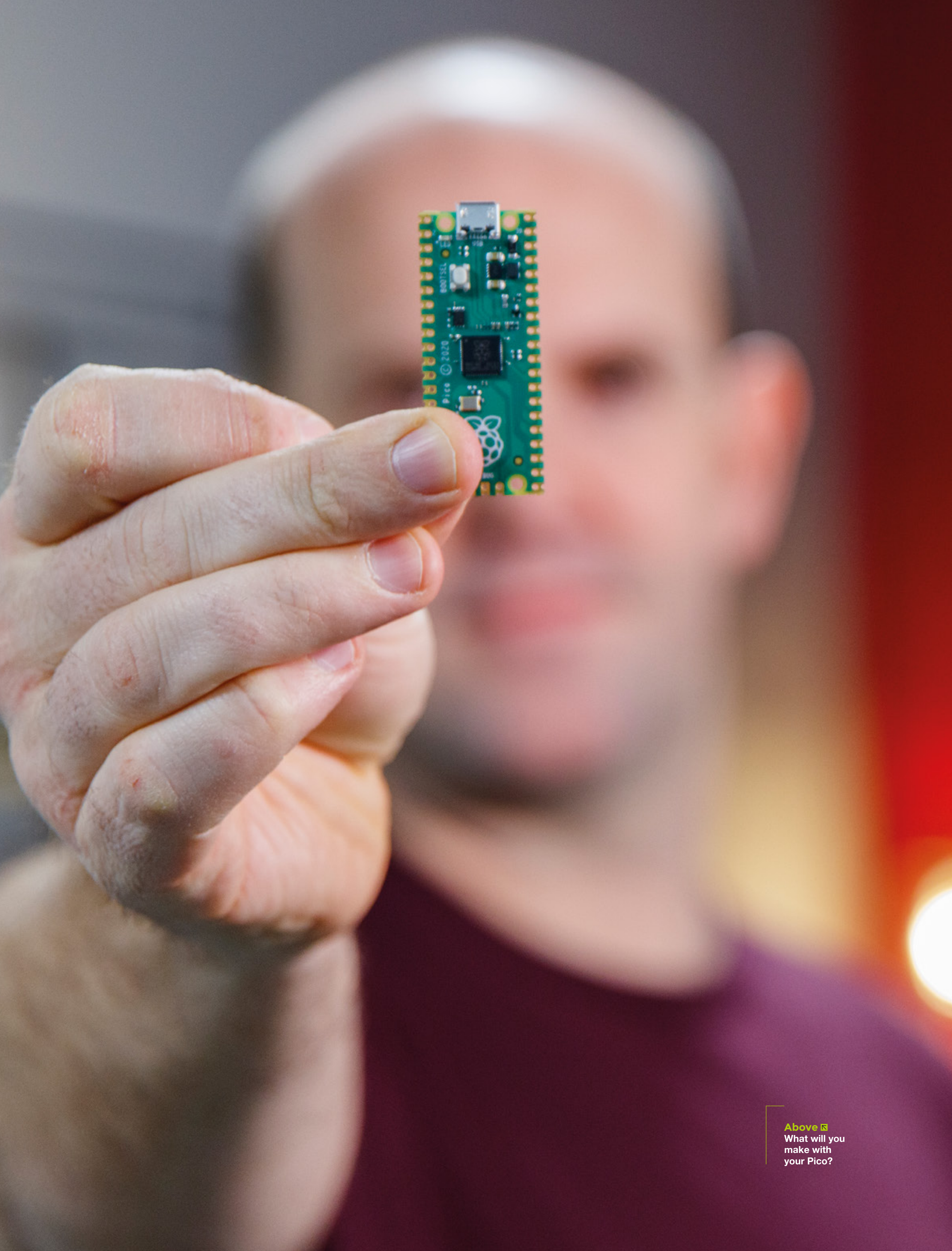
**HS When you say Pico was 'expensive' to create, how expensive are you talking?**

**EU** Somewhere in the region of £3–4 million to design the chip.

**HS But why? You can buy a bunch of chips off the shelf and design a board and have a great product.**

**EU** You want to make something perfect, and in this space, there's no way to make a perfect thing without making your own silicon. There's a reason why the other fruit company does this, right? It's not that they want to save money. They have margin to burn. If they thought they could make great Macs using Intel chips, they wouldn't make their own silicon just for the vanity. They do it because it's what it takes to make a great product, and this is the same thing.

That's the privilege of working at Raspberry Pi. You get to watch a great team do great engineering. And they've done it again with the Pico! □



**Above** What will you make with your Pico?

IMPROVISER'S TOOLBOX

# MAGNETS

Make your builds more 'attractive'

**Mayank Sharma**

@geekybodhi

Mayank is a Padawan maker with an irrational fear of drills. He likes to replicate electronic builds, and gets a kick out of hacking everyday objects creatively.

**Although it might not be very apparent, you are surrounded by magnets and everything, from the computer to your car, wouldn't function properly without them.** In fact, our entire planet is like one big magnet. The Earth's iron core is surrounded by hot molten iron that is moving around it, creating an electric current that generates a magnetic field around the planet. This gives the planet a magnetic north and a magnetic south which, as we all know, is what the needle on a compass points to.

Surprisingly, despite its scale, the Earth's magnetic field is extremely weak – hundreds of times weaker than a typical bar magnet. But still, several animals can sense it. For instance, birds and turtles are known to navigate using magnetic fields and, according to some fairly recent research, magnets can repel sharks and rays.

"ALTHOUGH MOST OF THE MAGNETS AROUND YOU ARE MADE OF IRON, THEY CAN BE MADE OF ANY MATERIAL AS LONG AS THEY HAVE UNPAIRED ELECTRONS"

Kids have always been fascinated with magnets, but physicists understand them pretty well, for the most part. So, for instance, scientists know that magnetism is the result of magnetic fields that naturally radiate from the electrically charged particles that make up the atoms. They also know that the most common magnetic fields originate from electrons, which are negatively charged.

In non-magnetic materials, the magnetic fields of electrons point in different directions and cancel each other out. In magnetic materials, the fields all align in the same direction. The physicists have multiple explanations about why the magnetic fields align in the same direction, which also helps explain how magnets behave.

What still remains a mystery though is why particles emit magnetic fields in the first place, and why do magnets always have a north and south pole?

You can cut a magnet as many times as you wish and the pieces will still have the two poles. However, hammering it or heating it up will cause it to lose its magnetic properties. That's because these actions cause the particles to lose their alignment and get arranged in random directions.

Although most of the magnets around you are made of iron, they can be made of any material as long as they have unpaired electrons. That includes many metals and alloys, such as neodymium.

In addition to naturally occurring magnets, we also have electromagnets that use electricity to create their power of magnetism, which can be turned off simply by killing the electricity. Powerful electromagnets are used in high-speed Maglev trains, to make the trains float over the tracks, reducing friction. The technology is also popularly used for propelling rollercoasters.

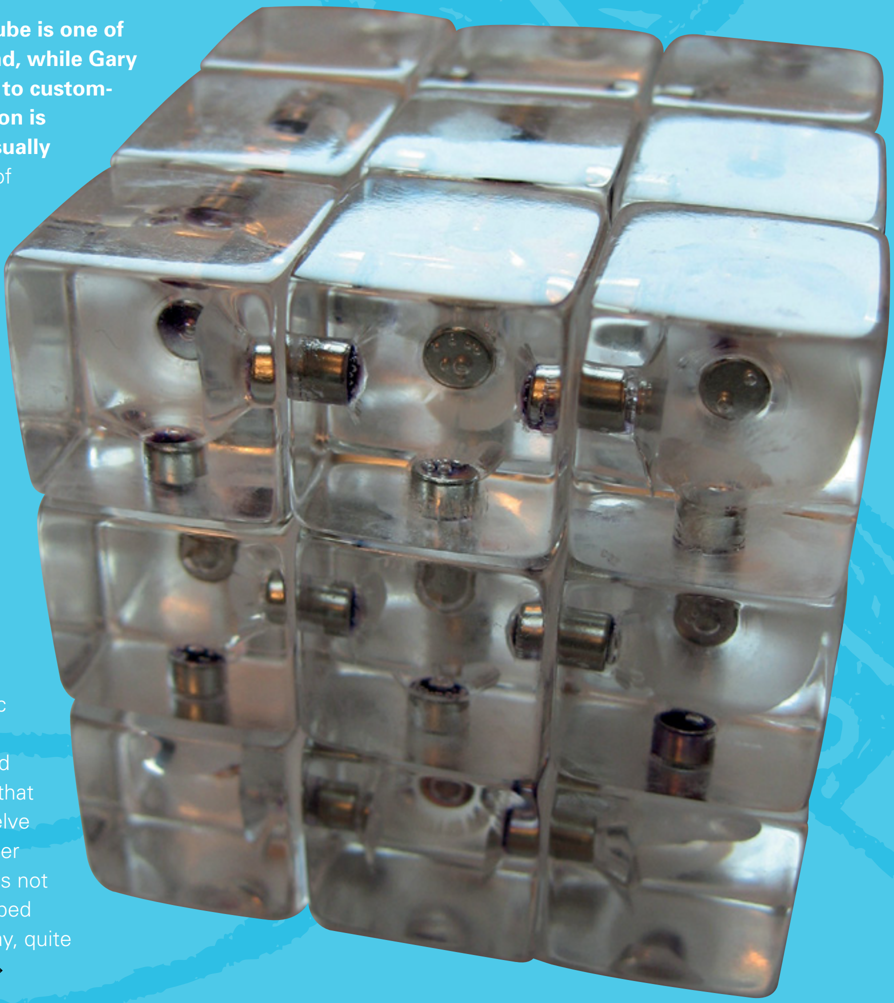
Here are some interesting builds that make clever use of magnets.

# RUBIK'S CUBE

**Project Maker**
**GARY FIXLER**

**Project Link**
hsmag.cc/RubiksCube

**W****e think a Rubik's Cube is one of the best puzzles and, while Gary Fixler isn't the first to custom-build one, his version is one of the most visually arresting.** Gary's magnetic version of the original Rubik's Cube is made with 27 acrylic cubes and, instead of screws through springs, uses 108 neodymium disc magnets. Gary explains that he first mocked up a version in the Maya 3D design app to figure out the polarities of the magnets, and he only began working on the project once he was satisfied his cube would remain sound, even as faces were spun all around. The project involves a lot of drilling and gluing, and Gary has patiently explained the entire process in great detail. Besides getting the polarities of all the magnets right, the toughest part was the intricate drilling in the acrylic cubes, which took some missteps before he got it right. He first created the central piece and the six pieces that attach to it, before attaching the twelve edge pieces, and the final eight corner pieces. "It was a lot of fun to build, is not nearly as easy to handle as a well-lubed Assembly Cube, but is in its own way, quite pretty and enjoyable," writes Gary. →



**Below** ↘
Although he prefers the clear cube, Gary suggests you should apply the colour labels to enjoy it like a real puzzle

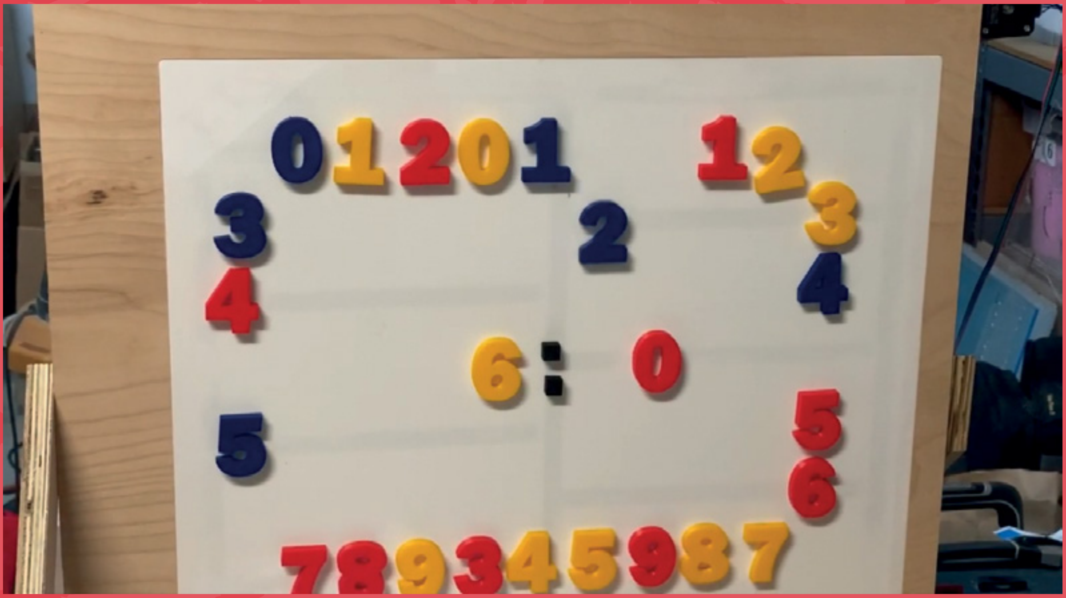
# ANIMATED CLOCK

**Project Maker**
**Craig Colvin**

**Project Link**
**hsmag.cc/MagnetClock**

**There's no shortage of custom DIY clocks, but Craig Colvin's version is a head-turner, and not just because it uses fridge magnets.** Craig places the magnetic numbers on a piece of thin white Plexiglas that's got a sheet of metal laminated to the back, to give the magnets something to hold on to. The real magic happens on the back though, where he's implemented a CoreXY-style mechanism to move the numbers. Craig explains his clock is driven by a belt that moves a carriage behind a number. It then engages two magnets which attract the magnets on the number, which allow the number to follow the carriage movement. Once the numbers are at their destination, the carriage magnets are disengaged and the metal sheet behind the Plexiglas keeps the numbers in place. The CoreXY assembly is powered by a stepper motor that's controlled by an Arduino-compatible, SAMD21 M0-Mini board from RobotDyn, along with a PCF8523 real-time clock for timing. The parts that hold the assembly are 3D-printed, and Craig has done a nice job of explaining the build process, including sharing the schematics of the electronics, and the source code for the various modules that make the whole contraption tick.

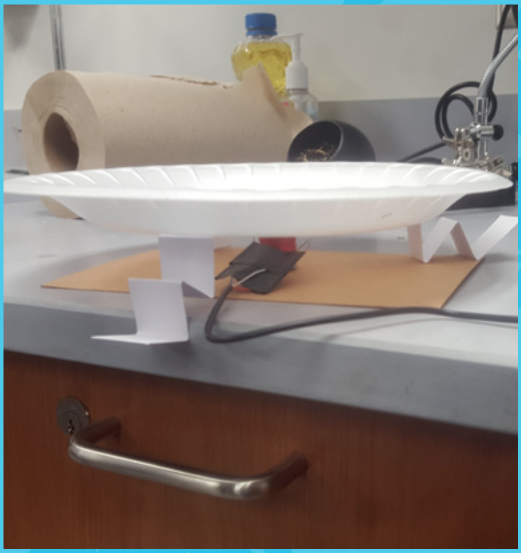
> "The parts that hold the assembly are 3D-printed"



**Right** ◆
**You can hack Craig's code and design to turn the clock into a spooky Halloween prop**

# SIMPLE SPEAKERS

**Speakers don't cost much these days, but the joy one gets from fabricating one from knick-knacks is priceless.** James built one for a workshop, to teach the fundamentals of these audio-oozing devices. The build requires a foam plate, 28-gauge magnetic wire to create the speaker coil, eight neodymium disc magnets, an aux cable with male 3.5 mm jacks on each end, and some other bits and bobs. The actual construction is fairly simple, and James's instructions are easy to follow. The only step that requires care is the last one, where you strip the aux cable to expose and identify the ground and signal wires. The speakers are very fragile and aren't very loud, but, as James demonstrates, they work really well and serve as a good way of demonstrating how speakers generate sound. James has also shared his notes about the workings of speakers for anyone interested in the science lesson.



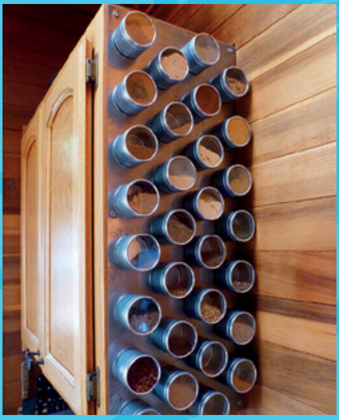
**Project Maker**
**JAMES**

**Project Link**
hsmag.cc/Speakers

**Left** ⬙
Although his speaker is mono, you can make it stereo by building another one and soldering the aux cable to both

# MAGNETIC SPICE RACK

**Back when prolific cookbook author Linda Ly was living in a 2000-square foot loft, she decided to free up some space in her kitchen by getting rid of the spinning spice rack.** Instead of storing the spices on the counter, Linda decided to mount them to the side of a cabinet with the help of some magnets. The build process is actually fairly simple. You'll need tin containers with clear, transparent lids. Make sure they are lightweight and big enough to store about three to five ounces of a spice. You'll also need neodymium magnets. If you get small ones, you can use two or even three per tin, depending on the weight of the spice. Linda's used printed labels to tag the name of the spice to the containers, but you can also scribble the name across the lid. Now glue the magnets to the back of the tins, fill the spices, and you're done. ☐



**Project Maker**
**LINDA LY**

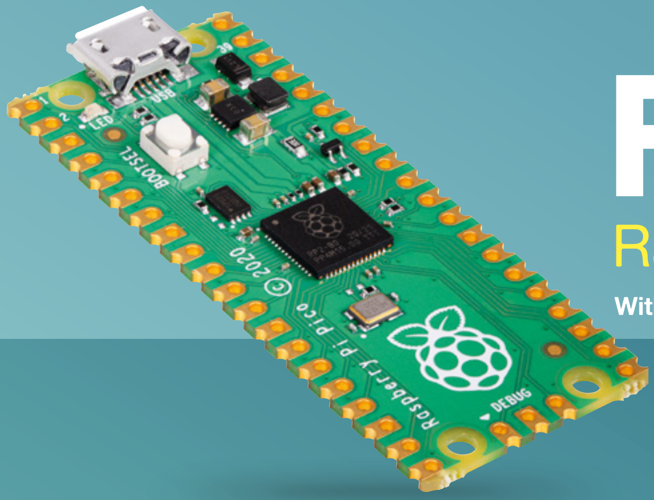
**Project Link**
hsmag.cc/SpiceRack

**Left** ⬙
Linda has mounted the containers on a steel sheet, but you could simply slap them to the side of your refrigerator as well

# SUBSCRIBE TODAY

Get 12 issues of HackSpace magazine delivered to your door for just

| | |
|---|---|
| **£55** (UK) | **£90** (USA) |
| **£80** (EU) | **£90** (Rest of World) |



## FREE!

Raspberry Pi Pico

**With your first 12-month print subscription**

This is a limited offer. Not included with renewals.
Offer subject to change or withdrawal at any time.

**Subscribe online:** hsmag.cc/subscribe

Subscription starts with next issue

# FORGE

HACK | MAKE | BUILD | CREATE

Improve your skills, learn something new, or just have fun tinkering – we hope you enjoy these hand-picked projects

PG 74

## RASPBERRY PI 400 PROJECTS

Great ways to use a keyboard computer

PG 78

## PARAMETRIC BOX

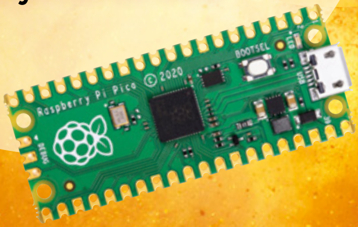
Use FreeCAD to create a customisable, 3D-printable project box

PG 68

## SCHOOL OF MAKING

Start your journey to craftsmanship with these essential skills

**68** **MicroPython on Pico**



PG 84

## SONIC CREATURE

Build a unique touch-sensitive instrument

PG 90

## NEOPIXEL DITHERING

Use Pico to get extra range from WS2812B LEDs

PG 96

## LASER CONTROLLER

Upgrade your laser cutter with a Raspberry Pi

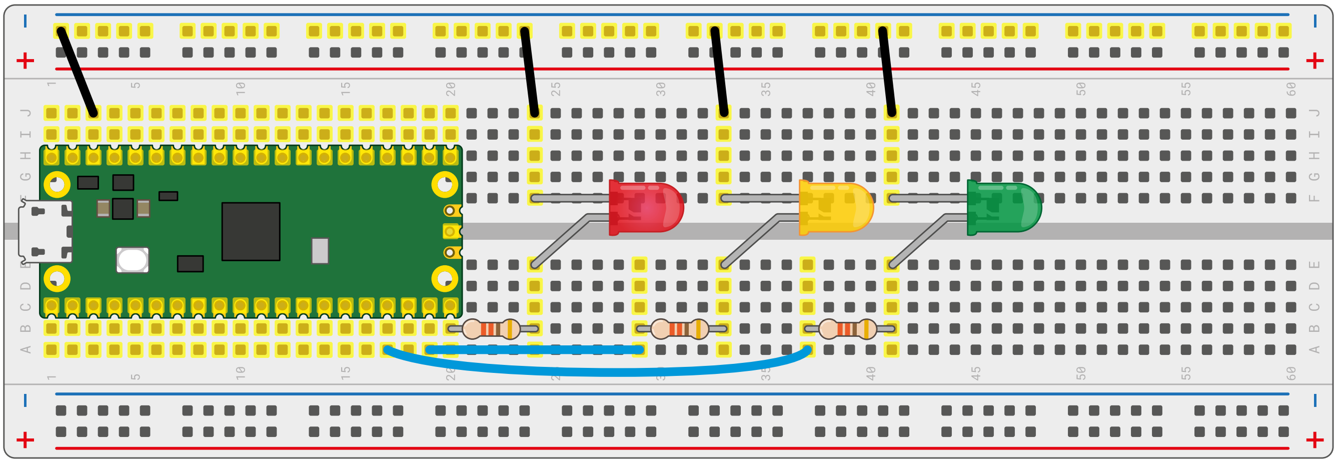


**Figure 1** ◈ A basic three-light traffic light system

# Traffic light controller

Create your own mini pedestrian crossing system using multiple LEDs and a push-button

**Gareth Halfacree**

@ghalfacree

Gareth Halfacree is a freelance technology journalist, writer, and former system administrator in the education sector. He has written several publications about Raspberry Pi.

**Microcontrollers can be found in almost all the electronic items you use on a daily basis – including traffic lights.** A traffic light controller is a specially built system which changes the lights on a timer, watches for pedestrians looking to cross, and can even adjust the timing of the lights depending on how much traffic there is – talking to nearby traffic light systems to ensure the whole traffic network keeps flowing smoothly.

While building a large-scale traffic management system is a pretty advanced project, it's simplicity itself to build a miniature simulator powered by your Raspberry Pi Pico. With this project, you'll see how to control multiple LEDs, set different timings, and how to monitor a push-button input while the rest of the program continues to run using a technique known as threading.

For this project, you'll need your Pico; a breadboard; a red, a yellow or amber, and a green LED; three 330 Ω resistors; an active piezoelectric buzzer; and a selection of male-to-male (M2M) jumper wires. You'll also need a micro USB cable, and to connect your Pico to your Raspberry Pi or other computer running the Thonny MicroPython IDE.

## A SIMPLE TRAFFIC LIGHT

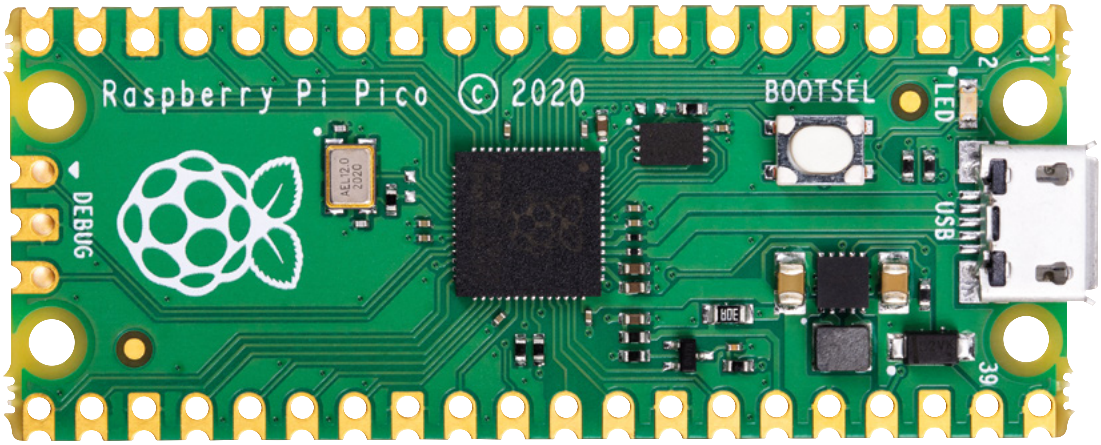
Start by building a simple traffic light system, as shown in **Figure 1**. Take your red LED and insert it into the breadboard so it straddles the centre divide. Use one of the 330 Ω resistors, and a jumper wire if you need to make a longer connection, to connect the longer leg – the anode – of the LED to the pin at the bottom-left of your Pico as seen from the top with the micro USB cable uppermost, GP15. If you're using a numbered breadboard and have your Pico inserted at the very top, this will be breadboard row 20.

Take a jumper wire and connect the shorter leg – the cathode – of the red LED to your breadboard's ground rail. Take another, and connect the ground rail to one of your Pico's ground (GND) pins – in the **Figure 1** wiring diagram, we've used the ground pin on row three of the breadboard.

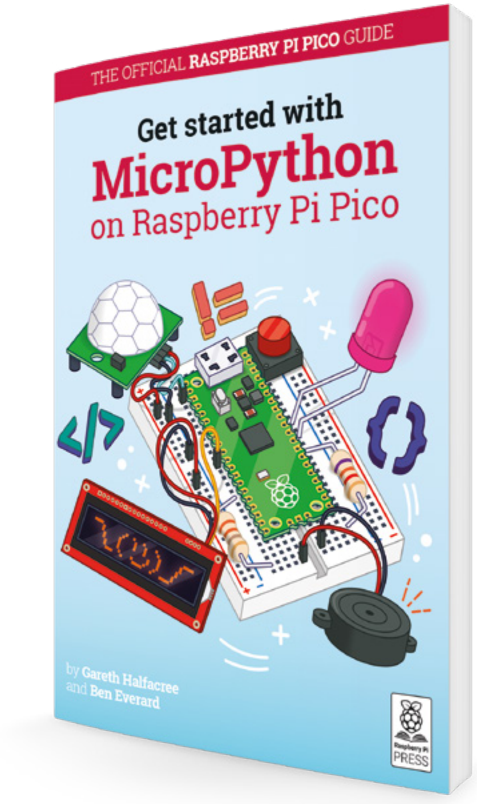
You've now got one LED connected to your Pico, but a real traffic light has at least two more for a total of three: a red light to tell the traffic to stop, an amber or yellow light to tell the traffic the light is about to change, and a green LED to tell the traffic it can go again.

Take your amber or yellow LED and wire it to your Pico in the same way as the red LED, making sure the shorter leg is the one connecting to the ground rail of the breadboard and that you've got the 330 Ω resistor in place to protect it. This time, though, wire the longer leg – via the resistor – to the pin next to the one to which you wired the red LED, GP14.

Finally, take the green LED and wire it up the same way again – remembering the 330 Ω resistor – to pin GP13. This isn't the pin right next to pin GP14, though – that pin is a ground (GND) pin, which you can see if you look closely at your Pico: the ground pins all have



**Left** Raspberry Pi Pico is a brand new microcontroller



**Below** This article is an extract from *Get Started with MicroPython on Raspberry Pi Pico.* Buy it in print or download the PDF: hsmag.cc/picobook

a square shape to their pads, while the other pins are round.

When you've finished, your circuit should match **Figure 1**: a red, a yellow or amber, and a green LED, all wired to different GPIO pins on your Pico via individual 330 Ω resistors and connected to a shared ground pin via your breadboard's ground rail.

To program your traffic lights, connect your Pico to your Raspberry Pi or other computer and load Thonny. Create a new program, and start by importing the machine library so you can control your Pico's GPIO pins:

```
import machine
```

You'll also need to import the utime library, so you can add delays between the lights going on and off:

```
import utime
```

As with any program using your Pico's GPIO pins, you'll need to set each pin up before you can control it:

```
led_red = machine.Pin(15, machine.Pin.OUT)
led_amber = machine.Pin(14, machine.Pin.OUT)
led_green = machine.Pin(13, machine.Pin.OUT)
```

These lines set pins GP15, GP14, and GP13 up as outputs, and each is given a descriptive name to make it easier to read the code: 'led', so you know the pins control an LED, and then the colour of the LED.

## WARNING

Always remember that an LED needs a current-limiting resistor before it can be connected to your Pico. If you connect an LED without a current-limiting resistor in place, the best outcome is the LED will burn out and no longer work; the worst outcome is it could do the same to your Pico.

Real traffic lights don't run through once and stop – they keep going, even when there's no traffic there and everyone's asleep. So that your program does the same, you'll need to set up an infinite loop:

```
while True:
```

Each of the lines beneath this need to be indented by four spaces, so MicroPython knows they form →

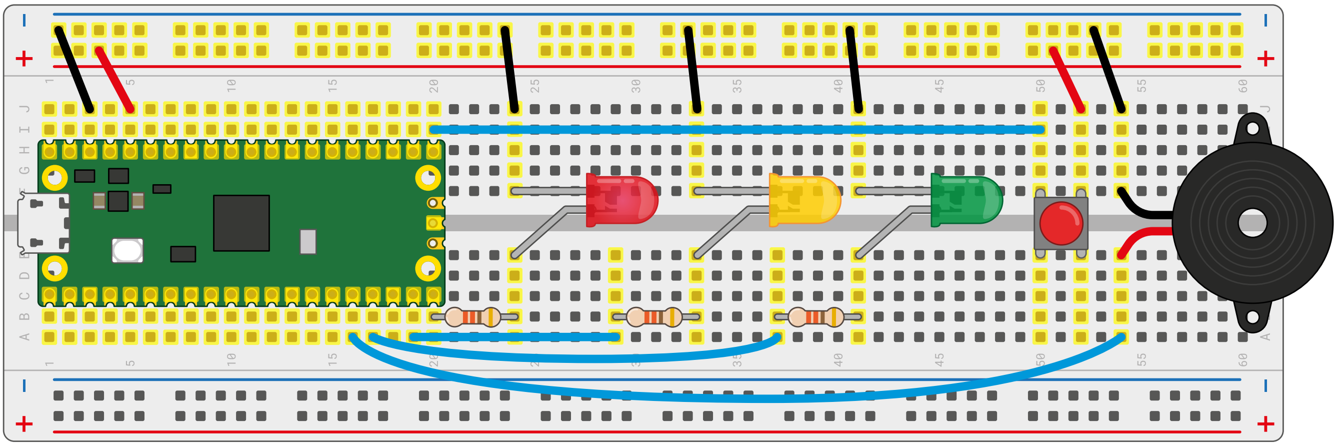


**Figure 2** A puffin crossing traffic light system

part of the loop; when you press the **ENTER** key Thonny will automatically indent the lines for you.

```
led_red.value(1)
utime.sleep(5)
led_amber.value(1)
utime.sleep(2)
led_red.value(0)
led_amber.value(0)
led_green.value(1)
utime.sleep(5)
led_green.value(0)
led_amber.value(1)
utime.sleep(5)
led_amber.value(0)
```

Click the Run icon and save your program to your Pico as **Traffic_Lights.py**. Watch the LEDs: first the red LED will light up, telling the traffic to stop; next, the amber LED will come on to warn drivers the lights are about to change; next, both LEDs switch off and the green LED comes on to let traffic know it can pass; then the green LED goes off and the amber one comes on to warn drivers the lights are about to change again; finally, the amber LED goes off – and the loop restarts from the beginning, with the red LED coming on.

The pattern will loop until you press the Stop button, because it forms an infinite loop. It's based on the traffic light pattern used in real-world traffic control systems in the UK and Ireland, but sped up – giving cars just five seconds to pass through the lights wouldn't let the traffic flow very freely!

**Right** Resistors are essential components for limiting the amount of current that flows through a circuit

Real traffic lights aren't just there for road vehicles, though: they are also there to protect pedestrians, giving them an opportunity to cross a busy road safely. In the UK, the most common type of these lights are known as pedestrian-operated user-friendly intelligent crossings or puffin crossings.

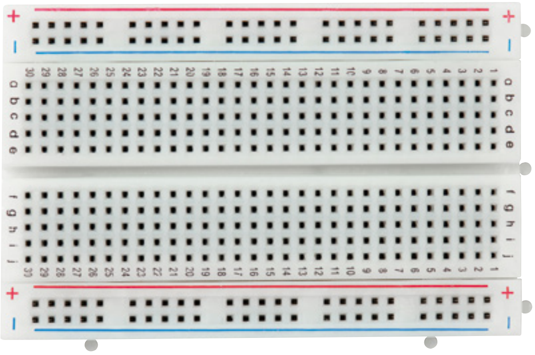
To turn your traffic lights into a puffin crossing, you'll need two things: a push-button switch, so the pedestrian can ask the lights to let them cross the road; and a buzzer, so the pedestrian knows when it's their turn to cross. Wire those into your breadboard as in **Figure 2**, with the switch wired to pin GP16 and the 3V3 rail of your breadboard, and the buzzer wired to pin GP12 and the ground rail of your breadboard.

> "An easy way to visualise threads is to think of each **one as a separate worker in a kitchen**"

If you run your program again, you'll find the button and buzzer do nothing. That's because you haven't yet told your program how to use them. In Thonny, go back to the lines where you set up your LEDs and add the following two new lines below:

```
button = machine.Pin(16, machine.Pin.IN)
buzzer = machine.Pin(12, machine.Pin.OUT)
```

This sets the button on pin GP16 up as an input, and the buzzer on pin GP12 as an output. Remember, your Pico has built-in programmable resistors for its inputs, which run as pull-down resistors by default; if you were writing this program for a different microcontroller development board, you might need



to configure the programmable resistors manually – or even place a physical resistor on your breadboard.

Next, you need a way for your program to constantly monitor the value of the button. Previously, all your programs have worked step-by-step through a list of instructions – only ever doing one thing at a time. Your traffic light program is no different: as it runs, MicroPython walks through your instructions step-by-step, turning the LEDs on and off.

For a basic set of traffic lights, that's enough; for a puffin crossing, though, your program needs to be able to record whether the button has been pressed in a way that doesn't interrupt the traffic lights. To make that work, you'll need a new library: _thread. Go back to the section of your program where you import the machine and utime libraries, and import the _thread library:

```
import _thread
```

A thread or thread of execution is, effectively, a small and partially independent program. You can think of the loop you wrote earlier, which controls the lights, as the main thread of your program – and using the _thread library you can create an additional thread, running at the same time.

An easy way to visualise threads is to think of each one as a separate worker in a kitchen: while the chef is preparing the main dish, someone else is working on a sauce. At the moment, your program has only one thread – the one which controls the traffic lights. The RP2040 microcontroller which powers your Pico, however, has two processing cores – meaning, like the chef and the sous-chef in the kitchen, you can run two threads at the same time to get more work done.

Before you can make another thread, you'll need a way for the new thread to pass information back to the main thread – and you can do this using global variables. The variables you've been working with prior to this are known as local variables, and only work in one section of your program; a global variable works everywhere, meaning one thread can change the value and another can check to see if it has been changed.

To start, you need to create a global variable. Below your `buzzer =` line, add the following:

```
global button_pressed
button_pressed = False
```

This sets up `button_pressed` as a global variable, and gives it a default value of `False` – meaning when the program starts, the button hasn't yet been pushed. The next step is to define your thread, by adding the following lines directly below – adding a blank line, if you want, to make your program more readable:

```
def button_reader_thread():
    global button_pressed
    while True:
        if button.value() == 1:
            button_pressed = True
```

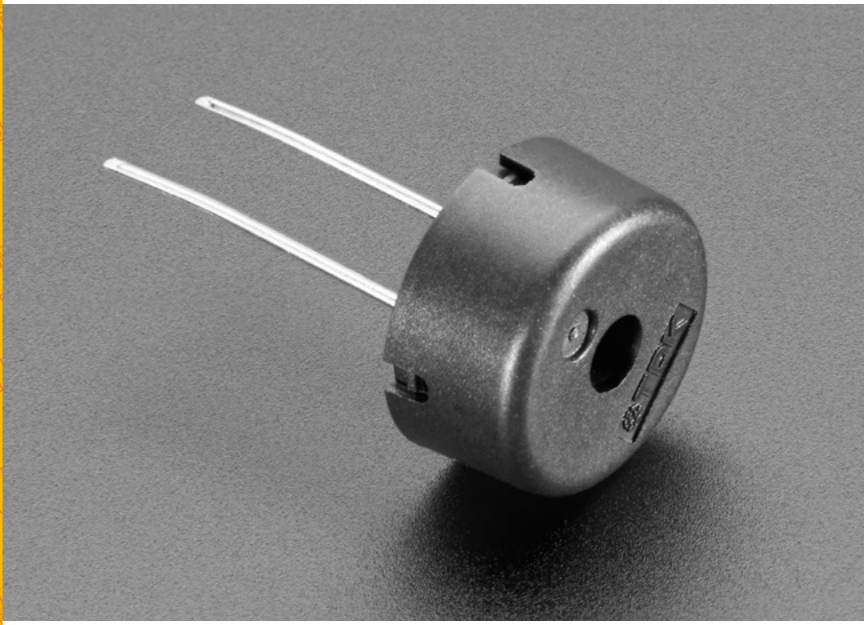
The first line you've added defines your thread and gives it a name which describes its purpose: a thread to read the button input. Like when writing a loop, MicroPython needs everything contained within the thread to be indented by four spaces – so it knows where the thread begins and ends.

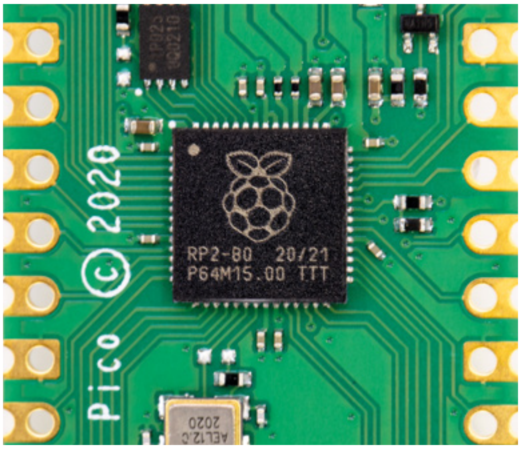
The next line lets MicroPython know you're going to be changing the value of the global `button_pressed` variable. If you only want to check the value, you wouldn't need this line – but without it you can't make any changes to the variable.

Next, you've set up a new loop – which means a new four-space indent needs to follow, for eight in →

**Left**
Breadboards make it easy to prototype circuits

**Above**
LEDs come in different colours and are a great way to add light to your project





**Above** There are many ways of making noise with a microcontroller, but a buzzer is the simplest

**Right** The RP2040 microcontroller at the heart of Pico has two ARM Cortex-M0 cores

total, so MicroPython knows both that the loop is part of the thread and the code below is part of the loop. This nesting of code in multiple levels of indentation is very common in MicroPython, and Thonny will do its best to help you by automatically adding a new level each time it's needed – but it's up to you to remember to delete the spaces it adds when you're finished with a particular section of the program.

The next line is a conditional which checks to see if the value of the button is 1. Because your Pico uses an internal pull-down resistor, when the button isn't being pressed the value read is 0 – meaning the code under the conditional never runs. Only when the button is pressed will the final line of your thread run: a line which sets the **`button_pressed`** variable to **`True`**, letting the rest of your program know the button has been pushed.

> Defining a thread doesn't set it running: it's possible **to start a thread at any point in your program**

You might notice there's nothing in the thread to reset the **`button_pressed`** variable back to **`False`** when the button is released after being pushed. There's a reason for that: while you can push the button of a puffin crossing at any time during the traffic light cycle, it only takes effect when the light has gone red and it's safe for you to cross. All your new thread needs to do is to change the variable when the button has been pushed; your main thread will handle resetting it back to **`False`** when the pedestrian has safely crossed the road.

Defining a thread doesn't set it running: it's possible to start a thread at any point in your program, and you'll need to specifically tell the _thread library when you want to launch the thread. Unlike running a normal line of code, running the thread doesn't stop the rest of the program: when the thread starts, MicroPython will carry on and run the next line of your program even as it runs the first line of your new thread.

Create a new line below your thread, deleting all of the indentation Thonny has automatically added for you, which reads:

```
_thread.start_new_thread(button_reader_thread, ())
```

This tells the _thread library to start the thread you defined earlier. At this point, the thread will start to run and quickly enter its loop – checking the button thousands of times a second to see if it's been pressed yet. The main thread, meanwhile, will carry on with the main part of your program.

Click the Run button now. You'll see the traffic lights carry on their pattern exactly as before, with no delay or pauses. If you press the button, though, nothing will happen – because you haven't added the code to actually react to the button yet.

Go to the start of your main loop, directly underneath the line **`while True:`**, and add the following code – remembering to pay attention to the nested indentation, and deleting the indentation Thonny has added when it's no longer required:

```
    if button_pressed == True:
        led_red.value(1)
        for i in range(10):
            buzzer.value(1)
            utime.sleep(0.2)
            buzzer.value(0)
            utime.sleep(0.2)
        global button_pressed
        button_pressed = False
```

**Above**
Your traffic light needs to light up in the correct sequence

This chunk of code checks the **`button_pressed`** global variable to see if the push-button switch has been pressed at any time since the loop last ran. If it has, as reported by the button reading thread you made earlier, it begins running a section of code which starts by turning the red LED on to stop traffic and then beeps the buzzer ten times – letting the pedestrian know it's time to cross.

Finally, the last two lines reset the **`button_pressed`** variable back to **`False`** – so the next time the loop runs it won't trigger the pedestrian crossing code unless the button has been pushed again. You'll see you didn't need the line **`global button_pressed`** to check the status of the variable in the conditional; it's only needed when you want to change the variable and have that change affect other parts of your program.

Your finished program should look like this:

```
import machine
import utime
import _thread

led_red = machine.Pin(15, machine.Pin.OUT)
led_amber = machine.Pin(14, machine.Pin.OUT)
led_green = machine.Pin(13, machine.Pin.OUT)
button = machine.Pin(16, machine.Pin.IN)
buzzer = machine.Pin(12, machine.Pin.OUT)

global button_pressed
button_pressed = False

def button_reader_thread():
    global button_pressed
    while True:
        if button.value() == 1:
            button_pressed = True

_thread.start_new_thread(button_reader_thread, ())

while True:
    if button_pressed == True:
        led_red.value(1)
        for i in range(10):
            buzzer.value(1)
            utime.sleep(0.2)
            buzzer.value(0)
            utime.sleep(0.2)
        global button_pressed
        button_pressed = False
    led_red.value(1)
    utime.sleep(5)
    led_amber.value(1)
    utime.sleep(2)
    led_red.value(0)
    led_amber.value(0)
    led_green.value(1)
    utime.sleep(5)
    led_green.value(0)
    led_amber.value(1)
    utime.sleep(5)
    led_amber.value(0)
```

Click the Run icon. At first, the program will run as normal: the traffic lights will go on and off in the usual pattern. Press the push-button switch: if the program is currently in the middle of its loop, nothing will happen until it reaches the end and loops back around again – at which point the light will go red and the buzzer will beep to let you know it's safe to cross the road.

The conditional section of code for crossing the road runs before the code you wrote earlier for turning the lights on and off in a cyclic pattern: after it's finished, the pattern will begin as usual with the red LED staying lit for a further five seconds on top of the time it was lit while the buzzer was going. This mimics how a real puffin crossing works: the red light remains lit even after the buzzer has stopped sounding, so anyone who started to cross the road while the buzzer was going has time to reach the other side before the traffic is allowed to go.

Let the traffic lights loop through their cycle a few more times, then press the button again to trigger another crossing. Congratulations: you've built your own puffin crossing!

# Perfect projects for your Raspberry Pi 400

It's a keyboard and a computer! The portability and compact form factor of the new Raspberry Pi 400 make for perfect project material

**PJ Evans**

@MrPJEvans

PJ is a writer, tinkerer, and software engineer. He keeps misplacing his Raspberry Pi 400 and then wondering why his keyboard won't boot.

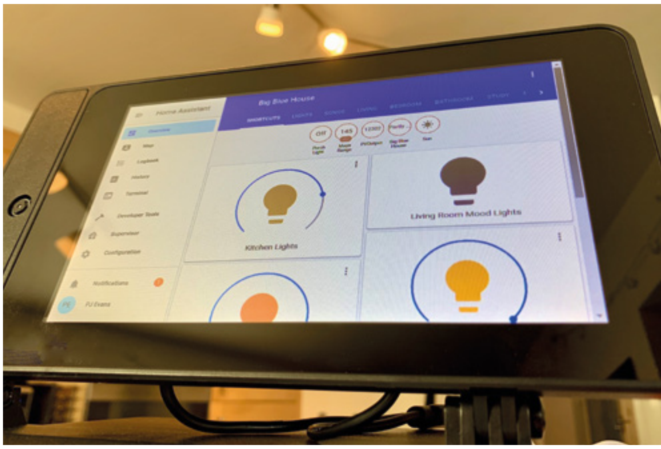
**You've probably guessed that we at HackSpace magazine are pretty excited about Raspberry Pi 400.** Having a desktop-capable computer in a single keyboard unit for under £100 is not only amazing, but also takes some of us older ones back to the time when this form factor was the norm. That said, does it deserve a place in the maker community? Obviously, the regular Raspberry Pi 'credit card' format is great for robotics and being nestled away out of sight. So what can makers do with the slightly larger 400? Well, there's plenty to keep you busy. Here's some inspiration.

## PROJECT 1
### RETRO COMPUTING CONSOLE

Let's start with one of the most popular uses for the Raspberry Pi range: classic gaming and computer emulation. Projects such as RetroPie (**retropie.org.uk**) and Lakka (**lakka.tv**) have made the otherwise complex setup of various console emulators an absolute breeze. A Raspberry Pi 400 is a great choice for emulating classic computers of the 1980s, such as the ZX Spectrum or BBC Micro (which after all a direct forerunner of the Raspberry Pi), as you have the computer and keyboard in a single portable unit. Plus, there's plenty of free and legal software available. Check out Fuse for a great Spectrum emulator.

## PROJECT 2
### HOME AUTOMATION

Again, the Raspberry Pi is a common choice for home automation projects. Combined with a small screen, a Raspberry Pi 400 can provide a compact console for controlling various aspects of your home. A popular platform is Home Assistant (**home-assistant.io**).



**Right** Home Assistant enables you to control all your smart devices from a single interface

You can download a ready-to-go image, burn it to an SD card, and be up and running in minutes. Home Assistant is compatible with a huge range of home automation and Internet of Things devices, including popular platforms like Philips Hue or IKEA TRÅDFRI for lighting, Sonos for audio, and sensor devices such as Nest. You can even build your own sensors using MQTT.

## PROJECT 3
### DESKTOP COMPUTER

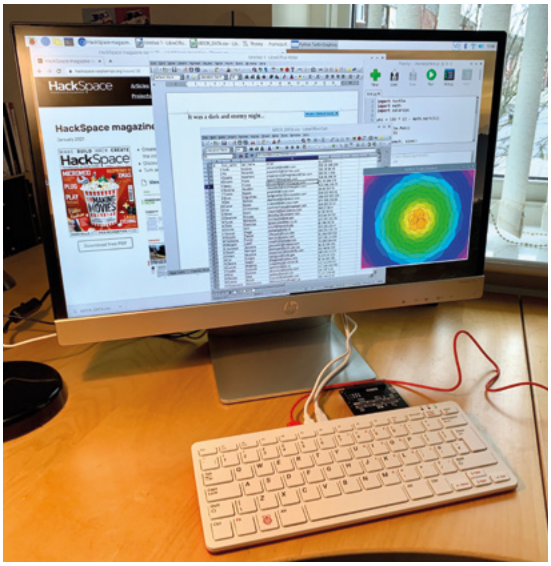
Raspberry Pi 400 is the fastest model yet. The addition of a large heatsink that runs almost the entire length of the keyboard has allowed the safe overclocking of the whole system, making it faster than Raspberry Pi 4. Add improvements in Chromium and you now have a realistic home computer that can be used for web browsing, YouTube, email and, with the amazing LibreOffice, a pretty decent machine for preparing documents, spreadsheets, and presentations. Of course, you're not going to be video editing or designing complex 3D models, but for casual use, you cannot beat the price.

## PROJECT 4
### SUPER-PORTABLE

One of the great pluses of Raspberry Pi 400 is that it reduces the number of wires needed, especially if you use a Bluetooth or wireless mouse. Better still, some of the more advanced power banks can provide enough amps to power the whole system. Suddenly, you've got a portable computer for making on the move. We've already seen projects for small touchscreens that connect to the GPIO pins and can be angled. Put all this together and you can set up your Raspberry Pi workstation anywhere and for a fraction of the cost of a laptop.

## PROJECT 5
### ZOOM STATION

Some of the most uttered phrases of 2020 must include "You're on mute" and "Can you see my screen?" For many of us, videoconferencing has become a daily way of life, not just with colleagues, but friends and family too. Why not set up a dedicated 'Zoom Station' so you can have meetings and chats in a more friendly environment? Just add a monitor, speaker, and webcam (many have decent microphones built-in). Raspberry Pi 400 has enough horsepower to happily run Zoom in the Chromium browser. With a big enough monitor, you can get the family around the table rather than squinting at a laptop screen. □



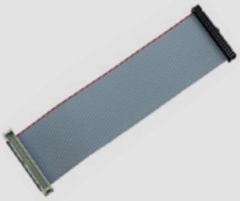
**Left** Raspberry Pi 400 is powerful enough to be a desktop computer

## ESSENTIAL ADD-ONS FOR RASPBERRY PI 400 MAKERS

### RIBBON CABLE

£3.50 / **thepihut.com**

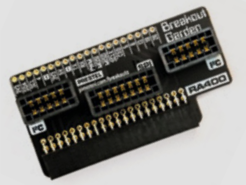
Make any HAT fit nicely to Raspberry Pi 400's GPIO port with the addition of a cheap 40×2 ribbon cable.



### BREAKOUT GARDEN

£10 / **shop.pimoroni.com**

Pimoroni's Breakout Gardens allow you to connect all kinds of sensors and output devices with ease. This three-slot version is right-angled for Raspberry Pi 400.



### WAVESHARE SCREEN

£54 / **thepihut.com**

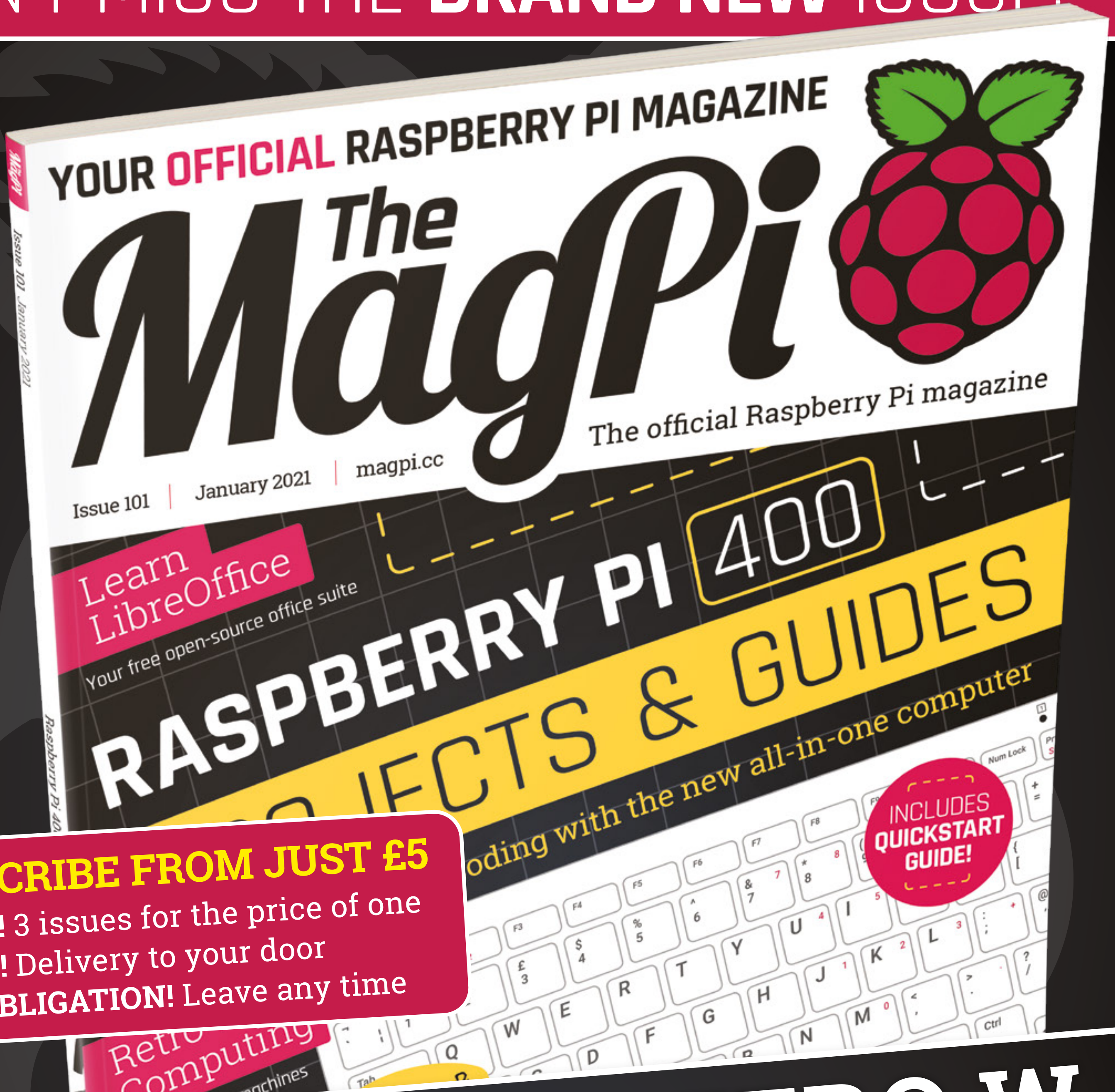
Need a perfect screen to go with your fancy new Raspberry Pi 400? If you're looking for portability, this 7" touchscreen comes with a case and connects to HDMI and USB.

### FLAT HAT HACKER

£7.50 / **shop.pimoroni.com**

Hooking the newly positioned GPIO port to a breadboard can be a bit fiddly. This breakout board allows you easier access to the pins, and can take an additional pHAT-format HAT as well.

DON'T MISS THE **BRAND NEW** ISSUE!


YOUR OFFICIAL RASPBERRY PI MAGAZINE
The MagPi
The official Raspberry Pi magazine
Issue 101 | January 2021 | magpi.cc
Learn LibreOffice
Your free open-source office suite
RASPBERRY PI 400
PROJECTS & GUIDES
...oding with the new all-in-one computer
INCLUDES QUICKSTART GUIDE!
Retro Computing
...machines
Issue 101 January 2021

**SUBSCRIBE FROM JUST £5**

- **FREE!** 3 issues for the price of one
- **FREE!** Delivery to your door
- **NO OBLIGATION!** Leave any time

WORTH £20

**FREE PI ZERO W STARTER KIT***

With your 12-month subscription to the print magazine

**magpi.cc/12months**

* While stocks last

**Buy online: store.rpipress.cc**

Antennas
Cellular, M2M
Zigbee/Thread
Sensors
Development Systems
Expansion Boards
Cables/Connectors
Wi-Fi
Digi-Key®
ELECTRONICS
LP-WAN
Bluetooth
Transceivers

The Building Blocks of IoT

Digi-Key is Your One Stop Shop for the Internet of Things (IoT) Components.

DIGIKEY.COM/IOT

9.6 MILLION+ PRODUCTS ONLINE | 1,200+ INDUSTRY-LEADING SUPPLIERS | 100% FRANCHISED DISTRIBUTOR

Digi-Key is a franchised distributor for all supplier partners. New products added daily. Digi-Key and Digi-Key Electronics are registered trademarks of Digi-Key Electronics in the U.S. and other countries. © 2021 Digi-Key Electronics, 701 Brooks Ave. South, Thief River Falls, MN 56701, USA

ECIA MEMBER
Supporting The Authorized Channel

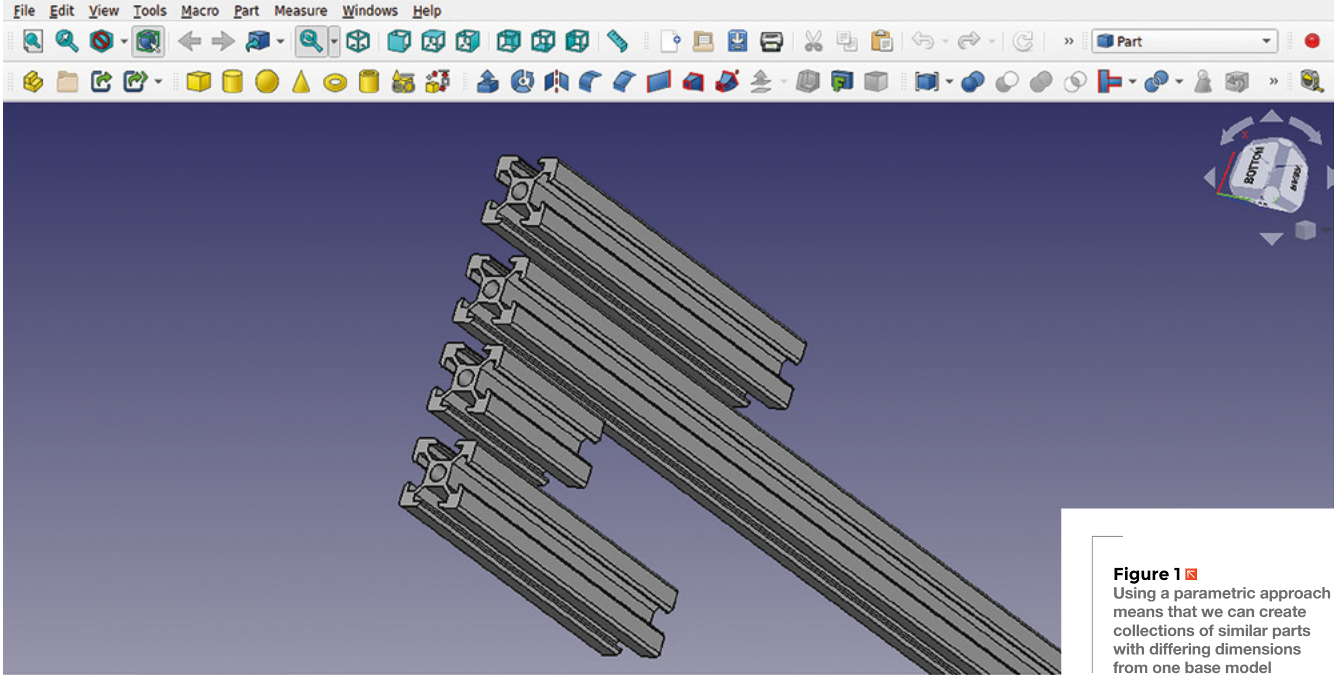


**Figure 1** Using a parametric approach means that we can create collections of similar parts with differing dimensions from one base model

# FreeCAD, parametric parts, and other approaches

In this third part of the FreeCAD series, we are going to design a variety of objects using numerous different tools and approaches

**Jo Hinchliffe**

@concreted0g

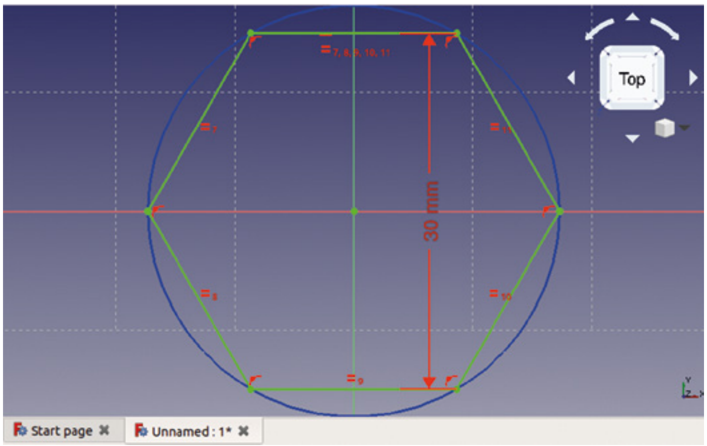
Jo Hinchliffe is a constant tinkerer and is passionate about all things DIY space. He loves designing and scratch-building both model and high-power rockets, and releases the designs and components as open-source. He also has a shed full of lathes and milling machines and CNC kit!

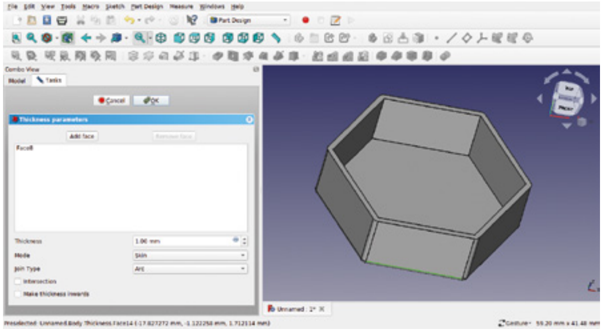
**In parts one and two of this series, we covered the basics of sketching, extruding, and other tasks to create parts using the part, sketcher, and part design workbenches.** In this tutorial, we're going to add to our skills by exploring a small range of tools and approaches and making a variety of small project items. We'll start by exploring the 'Add a Thickness' tool, look at using the Spreadsheet workbench, and working parametrically, we'll bring these skills altogether to make an automatic box with lid generator.

For a quick example of using the 'Make a Thick Solid' tool, begin in the part design workbench and create a new body. Create a new sketch in the body on the XY plane, then select the polygon tool to draw a hexagon. Start your hexagon on the 0,0 point of the axis so that it is positionally constrained. You should have a hexagon with two degrees of freedom. A simple way to constrain a hexagon is to make the uppermost line of the hexagon horizontal by selecting the line and clicking the 'create a horizontal constraint' tool. Next, you can select two nodes vertically above each other, and set a vertical distance to fully constrain the sketch **Figure 2**.

Close the sketch and then pad the sketch using the pad dialogue found on the tasks tab in the combo view window or by clicking the 'pad a selected sketch' tool icon.

We are going to now use the thickness tool to hollow out our hexagonal extrusion. Select the upper face of the hexagonal extrusion and then click the 'Make A Thick Solid' tool. You should see instantly that the part now appears as a hollowed version of itself. In the thickness tool dialog, you can see the selected face and change various parameters (**Figure 3**). You can, of course, increase and decrease the thickness that is created. By default, the thickness dialog adds the thickness to the outside of the underlying geometry, so if you set the hexagon vertical constraint to 30 mm you now will have an object that is 30 mm plus the thicknesses. If you





Face8

Thickness: 1.00 mm
Mode: Skin
Join Type: Arc
☐ Intersection
☐ Make thickness inwards

Preselected: Unnamed.Body.Thickness.Face14 (-17.827272 mm, -1.122250 mm, 1.712114 mm)

**Figure 2**
Constraining a hexagon centred on the 0,0 co-ordinates only needs two other constraints to fully constrain it

**Figure 3**
Using the 'Make a Thick Solid' tool to make our padded hexagon object hollow

want to create an object that matches the external dimensions of the underlying geometry, click the 'make thickness inwards' tick-box. You can also swap between Arc and Intersection, which essentially toggles between creating filleted edges or sharp edges on the thickness.

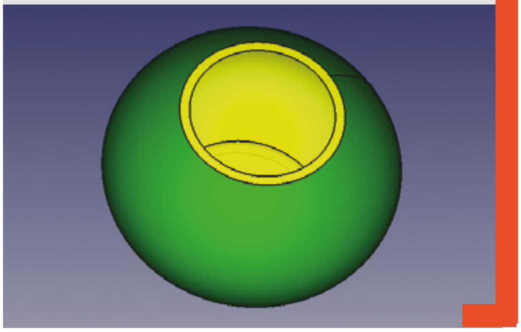
Finally, if you want to apply a thickness but create an object which is more of a pipe than a bowl, you can click the 'Add Face' button. In the preview window, the original selected face reappears highlighted and you can select the opposite face. You should now have a hexagonal object with a wall thickness with both ends open.

### LET'S GET PARAMETRIC!

We looked at making multiple 'simple copies' of parts using the 'part' menu on the part workbench in previous parts of the series. You can also use that approach to create simple copies of bodies that you have designed in part design. Sometimes we might want to make multiple similar parts, but with one or more dimensions changed – for example, if we were drawing a model of a 3D printer that used the common 2020 aluminium extrusion (**Figure 1**). One way we could make this easier is to draw an aluminium extrusion profile sketch, and then pad or extrude it multiple times to multiple lengths. As a workflow, we might choose to use the part design workbench to create a body for our original extrusion and then create multiple simple copies on the part workbench.

## OTHER OPTIONS

The thickness tool is an excellent way to quickly create hollow geometries, but it does have some limitations. It can fail to work with more complex shapes; things like curvy cones can be its nemesis. In the next part of this FreeCAD series, we'll begin to look at 'Lofting' tools, which enable us to create hollow parts with curved walls and more.



This is a very simple example that introduces the idea of working parametrically. Parametric work in CAD is where we can change the model geometry by adjusting parameters such as dimensions. Whilst only being one parameter, we can use this mini project to learn about using a spreadsheet to store and alter parameters for objects and sketches.

> **Parametric work in CAD is where we can change** the model geometry by adjusting parameters

We downloaded an image with the dimensions of the 2020 extrusion and created a new project. On the part design workbench we selected to create a new body and then to create a sketch in the XY plane. We crudely drew a shape of the profile using the polyline tool and then worked through to constrain the sketch (**Figure 4**).

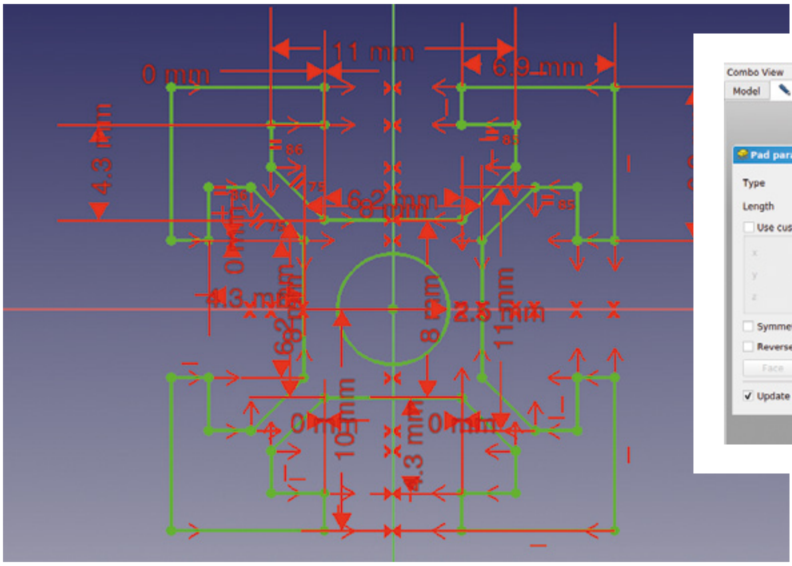
Having constrained the sketch, we checked how it looked by performing a 'pad' from the tasks menu, and we then added the external corner radius fillet and the chamfers at the edges of the extrusion slots. We could now, of course, just set a pad length of extrusion and then create a simple copy on the part workbench to create different length parts, but we →

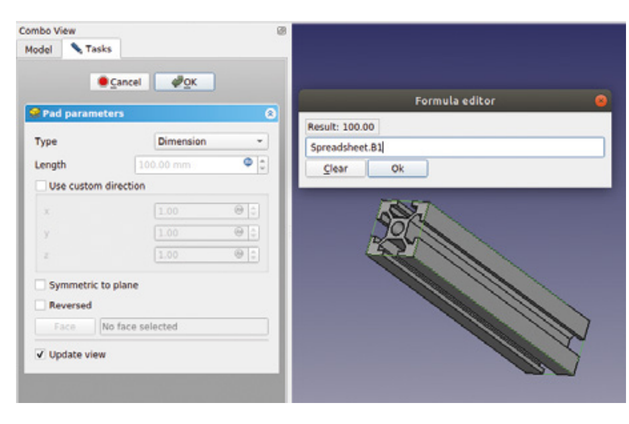
## QUICK TIP

Often in this article, we have described tools using the text that appears when you hover the mouse over the tool icon. This is a great way to find the tools or explore a new workbench.

## YOU'LL NEED

**A laptop or desktop computer**


11 mm
6.9 mm
0 mm
4.3 mm
2.5 mm
11 mm
8 mm
10 mm
4.3 mm
0 mm


Combo View
Model
Tasks
Cancel
OK
Pad parameters
Use custom direction
Symmetric to plane
Reversed
Face
No face selected
Update view
Formula editor
Clear
Ok

Type: Dimension

Length: 100.00 mm

Result: 100.00

Spreadsheet.B1

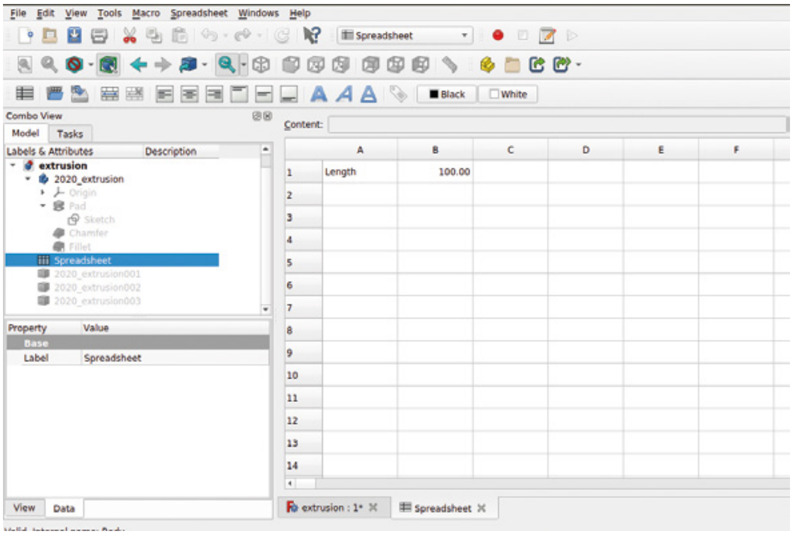
**Figure 4** A basic sketch of the 2020 profile. We added chamfers and fillets once the sketch was padded

**Figure 5** The spreadsheet workbench, with our spreadsheet opened in its own tab in the preview window

**Figure 6** Clicking the circular blue box in the corner of any input box takes you to the formula editor, where you can link the value to a spreadsheet, amongst other functions

want to learn how to use the spreadsheet function to further explore parametric work.

Let's move to the spreadsheet workbench using the workbench drop-down menu, and click the 'create a new spreadsheet' icon. You should now see a spreadsheet open in the preview window in a separate tab (**Figure 5**). This is very handy, as it allows quick switching between the model and the spreadsheet. If you close the spreadsheet tab, the spreadsheet is still an item in the file tree view and you can double-click to open it. The spreadsheet functions are very capable, and similar to standard spreadsheets you may have used in office software. For this simple task, we are going to write the label 'Length' in cell A1, and then in cell B1 we can type a length we want our extrusion to be. When we type a dimension into a constraint in the sketcher, it interprets that number as the units you use throughout FreeCAD, which can be set in the preferences menu. In our case, this is millimetres and will be the case for our spreadsheet values, so we don't need to define our input units in the spreadsheet.

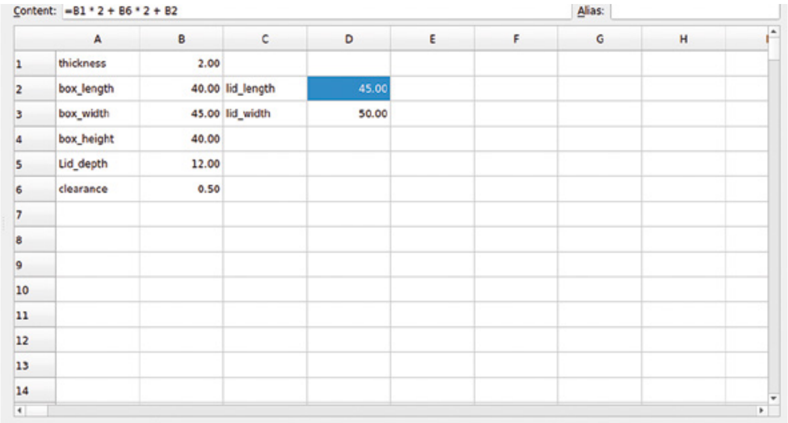

File
Edit
View
Tools
Macro
Spreadsheet
Windows
Help
Spreadsheet
Black
White
Combo View
Model
Tasks
Labels & Attributes
Description
extrusion
2020_extrusion
Origin
Pad
Sketch
Chamfer
Fillet
Spreadsheet
2020_extrusion001
2020_extrusion002
2020_extrusion003
Property
Value
Base
Label
Spreadsheet
View
Data
Content:
extrusion : 1*
Spreadsheet

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Length | 100.00 | | | | |

Set cell B1 to 100 and click back onto the preview tab with the model in it. Reopen the pad by double-clicking on the pad in the file tree. In the input box for 'length' you should see the current pad length value, but you should also see a blue circle icon on the right-hand side of the box . Click this to open the formula editor (**Figure 6**). In this input box, now start to type 'Spreadsheet'. You should find that it automatically suggests the word spreadsheet as you start typing. Click the suggested 'Spreadsheet' from the drop-down and it should insert the word spreadsheet followed by a full stop. After the full stop, we need to input the cell location of the cell from which we want it to take its value. If you type 'B', it again should automatically suggest 'B1' as a value. This autosuggestion is useful when we come to projects with numerous cells of data in a spreadsheet, as it will only suggest cells that have data in them. Click the 'B1' so that the value in the formula editor reads 'Spreadsheet.B1', as in **Figure 6**.

> **"This autosuggestion is useful when we come to projects** with numerous cells of data in a spreadsheet"

Returning to the preview tab with the model in, you should see that the extrusion body is now padded or extruded to the 100 mm length. You can now create multiple simple copies from the body using the 'create a simple copy' tool in the 'part' drop-down on the Part Workbench, changing the length of the original body copy in the spreadsheet, as in **Figure 1**.



Content: =B1 * 2 + B6 * 2 + B2

Alias:

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | thickness | 2.00 | | | | | | |
| 2 | box_length | 40.00 | lid_length | 45.00 | | | | |
| 3 | box_width | 45.00 | lid_width | 50.00 | | | | |
| 4 | box_height | 40.00 | | | | | | |
| 5 | Lid_depth | 12.00 | | | | | | |
| 6 | clearance | 0.50 | | | | | | |
| 7 | | | | | | | | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |
| 11 | | | | | | | | |
| 12 | | | | | | | | |
| 13 | | | | | | | | |
| 14 | | | | | | | | |

**Figure 7** The spreadsheet set up for our box generator

**Figure 8** Using the spreadsheet and the thickness tool, we have a parametric hollow box ready to make a lid for

## LET'S COMBINE OUR NEW SKILLS!

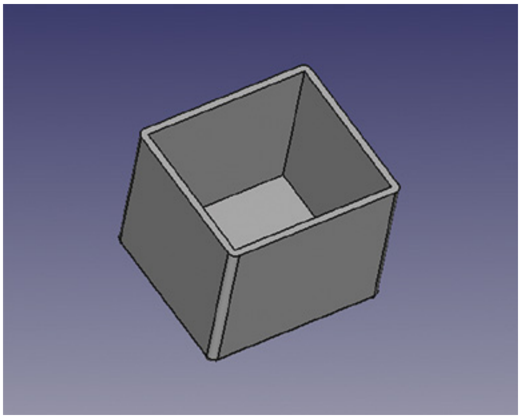
Having explored using both spreadsheets and the thickness tool, let's combine our new skills and create an automatic parametric box generator. This simple project makes it easy to create a box and lid of any size that we could 3D-print or CNC-machine.

Let's begin by creating a new project. In the new project, go to the spreadsheet workbench and create a new spreadsheet. For the main body of our box, we are going to need four parameters: the box length, width, height, and the wall thickness. Create labels for these parameters in cells, and in adjacent cells input some initial values (**Figure 7**).

Next, jump to the part design workbench and create a body and a sketch in the XY plane. We are going to draw a rectangle around the zero point and then, at first, constrain it to always be centred around the origin point of the sketch. Draw the rectangle and then click the 'create a symmetry' constraint tool. Use the symmetry constraint tool to select the upper right-hand node, the upper left-hand node, and the vertical Y axis line. Repeat this for the upper right and lower right nodes, and the X axis line. Now the rectangle should always be centred around the 0,0 co-ordinates, regardless of its size.

Next, let's add a constraint for the box length by selecting the top line and clicking the 'Fix the horizontal distance' tool. In the input box, click the blue circle formula editor button and insert 'Spreadsheet', followed by the cell location for the box length, which in our case was B2. Repeat this for the vertical line to set the box width, and the sketch should now appear fully constrained.

Close the sketch and perform a 'pad' task to extrude the box; set the pad dimension to receive the input from the box height cell in the spreadsheet. Next, we will click the upper surface of our box and use the thickness tool we explored earlier to hollow out our box. To set the thickness, we are going to link the thickness value in our spreadsheet. All being well, you should now have a hollow box in the preview window (**Figure 8**).



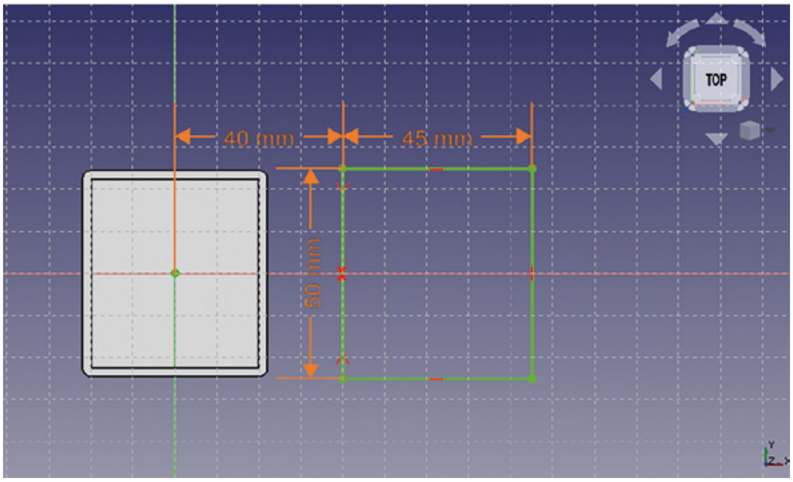


**Figure 9** Laying out the sketch for the box lid, making sure it will always be positioned away from our box

To make a lid, we are going to do the same process as we did to make the box section, but we are going to use some simple formulae so that the lid is generated to fit whatever dimensions are of the box created. On the part design workbench, click to create a new sketch on the XY plane. We don't need to import any geometry from our first sketch, but let's draw another rectangle and place it along the x axis so it isn't on top of our box base. To positionally →

## QUICK TIP

You don't have to draw the 2020 extrusion to learn this technique – you can just draw a simple square instead!

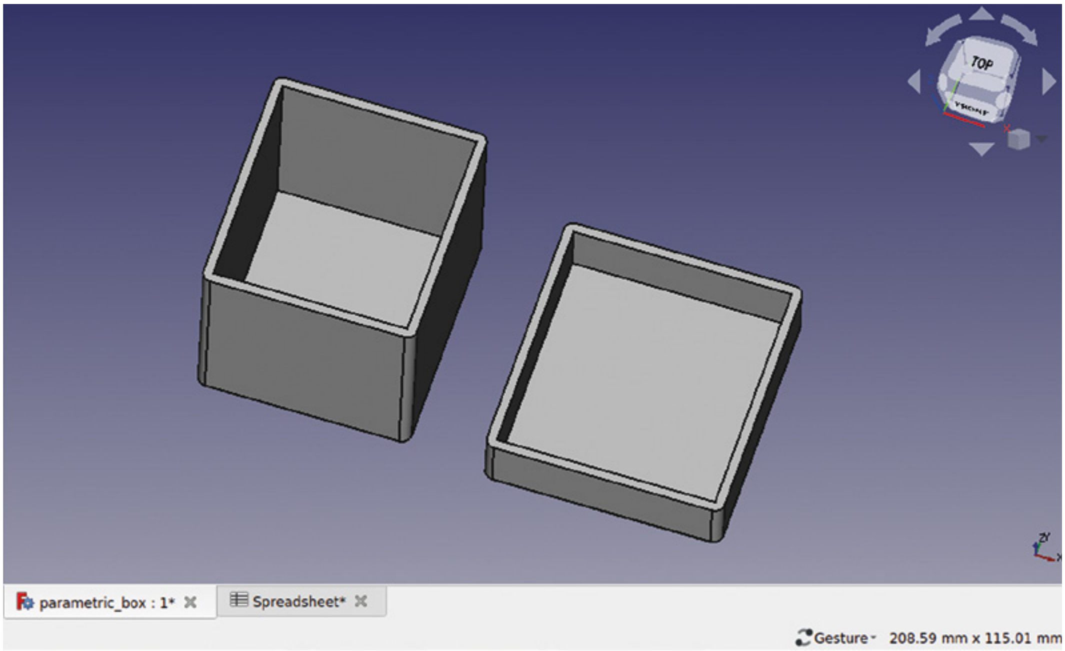
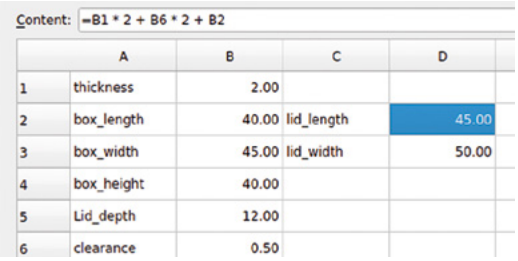


**Figure 11** The box now complete with a fitting lid laid out on the XY plane



Content: =B1 * 2 + B6 * 2 + B2

| | A | B | C | D |
|---|---|---|---|---|
| 1 | thickness | 2.00 | | |
| 2 | box_length | 40.00 | lid_length | 45.00 |
| 3 | box_width | 45.00 | lid_width | 50.00 |
| 4 | box_height | 40.00 | | |
| 5 | Lid_depth | 12.00 | | |
| 6 | clearance | 0.50 | | |

**Figure 10** Adding a small formula to a cell to automatically generate the dimension needed for the lid length

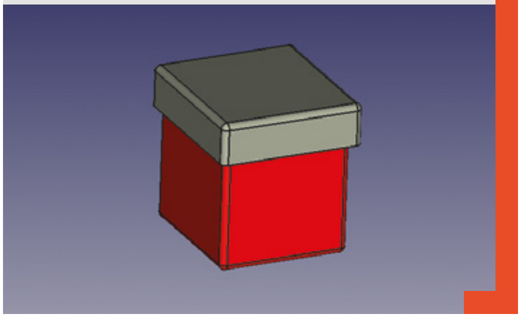
constrain this lid rectangle, select the upper left corner node and the sketch origin point at 0,0 and set a horizontal distance constraint. For the value of this constraint, we have linked the spreadsheet cell that holds the box_length value. This means that the lid will always be away from the box with a gap in between them (**Figure 9**).

Next, select the upper right and lower right nodes and the x axis line, and create a symmetry constraint. To create the length and width of the lid, we are going to create two new labels in the spreadsheet and create two new cells to hold this value. However, we are going to insert a small formula in the cells to generate these values. The value for the length of the lid will be equal to the box length value plus twice the value of the wall thickness. It will also need a clearance to make the lid fit over the box. We have created a label and a cell called 'clearance', and put in a value of half a millimetre which we will add to each side of the lid. So, to create the value of the lid_length in that cell, we input the formula to multiply the thickness and the clearance by two and add them to the box length. To add this formula, highlight the target cell and type it into the formula bar (**Figure 10**).

To finish the lid, we added the formula for the box width, using the same formula for length, except using the box width dimension. We also created an input value for the lid height. We constrained the length and width of the lid sketch by adding constraints linked to our calculated dimensions in the spreadsheet. We then performed a pad using the lid_depth value and, in turn, applied a thickness to the lid using the thickness value in the spreadsheet. You now have an automatic box generator at your disposal! Simply change the parameters in the spreadsheet and it will automatically generate your box with a fitting lid (**Figure 11**). □

## LINING UP

Once you have your box generator set up, you can generate boxes of any dimensions easily. The layout on the XY plane means that the box and lid will be easy to export and 3D-print, as both parts are modelled in a way that emulates them sitting on the print bed. If you wanted to look at your box with the lid in position, again you could create a simple copy on the part workbench and move and rotate those parts into the correct position.

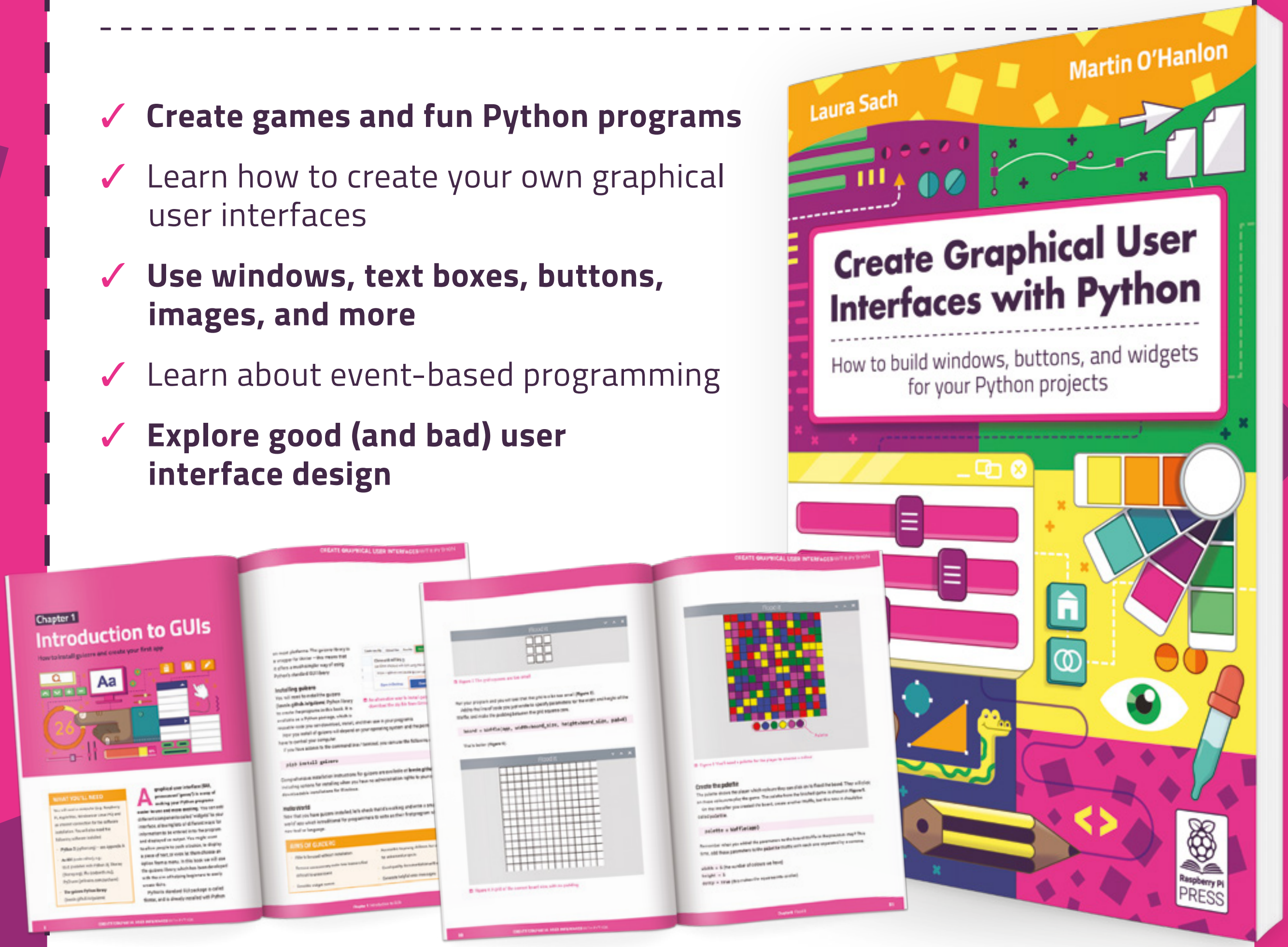


## QUICK TIP

When working on a sketch like the 2020 extrusion, using the 'create an equality' constraint tool is helpful on lines that are relatively the same in each quadrant of the sketch.

# Create Graphical User Interfaces with Python

- ✓ **Create games and fun Python programs**
- ✓ Learn how to create your own graphical user interfaces
- ✓ **Use windows, text boxes, buttons, images, and more**
- ✓ Learn about event-based programming
- ✓ **Explore good (and bad) user interface design**



Laura Sach
Martin O'Hanlon
Create Graphical User Interfaces with Python
How to build windows, buttons, and widgets for your Python projects
Raspberry Pi PRESS
Chapter 1
Introduction to GUIs

**Buy online:** hsmag.cc/pythongui

# Make music with touch

Use capacitive sensing to create your own sounds

**Helen Leigh**

@helenleigh

Helen turns all sorts of unusual items into musical instruments (among other things).

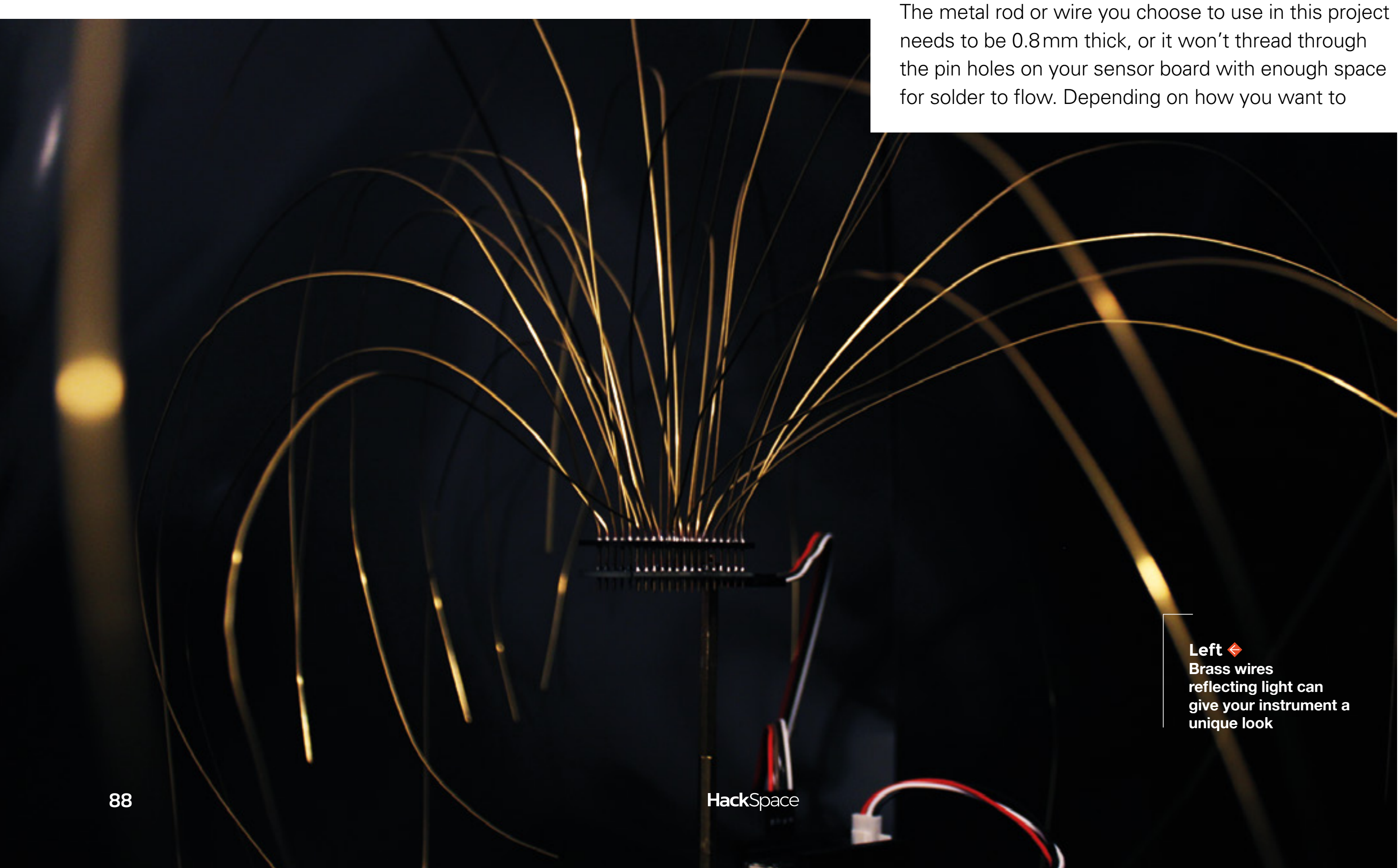
**At the heart of this make is one of my favourite board and sensor combinations: the Bela Mini and the Trill Craft.** It's a pretty pricey board compared to an entry-level microcontroller, but if you're looking to level up your embedded instrument design, it is really worth considering. If you're not ready to invest in a Bela, you can try out the Trill Craft sensor with an Arduino, Teensy, Raspberry Pi, or any other board that supports I2C.

The Trill Craft is a capacitive touch sensor board. Capacitive touch is a staple technology of many instrument inventors, many of whom will use a capacitive touch library or the trusty MPR121. The Trill Craft has great resolution and provides a continuous reading, plus the killer feature is that it has 30 channels, compared to the MPR121's twelve channels. You can also chain up to nine sensors together, giving you a potential 270 capacitive touch channels to play with.

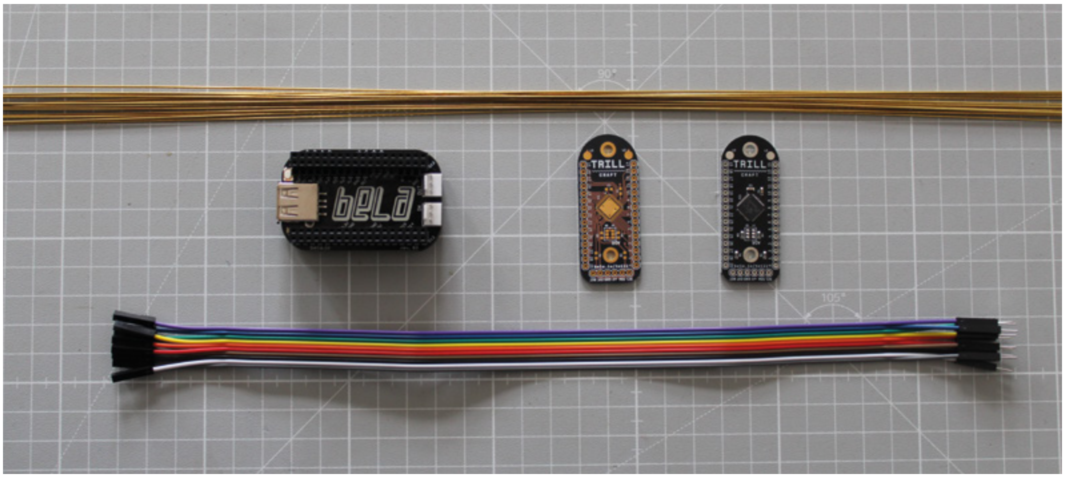
This project involves quite a few design decisions: limbs, body, shape, support, and strain relief. Take time making your choices and picking your materials – it will be worth it to create your own playable work of art.

## STEP ONE CHOOSING YOUR MATERIALS

The metal rod or wire you choose to use in this project needs to be 0.8 mm thick, or it won't thread through the pin holes on your sensor board with enough space for solder to flow. Depending on how you want to



**Left**
Brass wires reflecting light can give your instrument a unique look



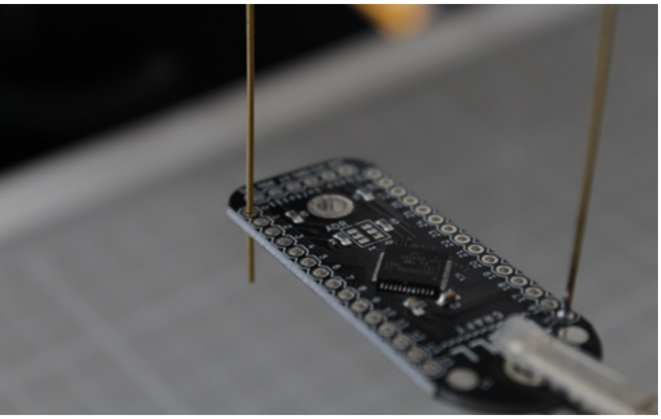
**Left**
The parts you'll need for this project

**Below**
The solder joints are critical, so take your time on them

shape the limbs of your creature, you'll also need it to be reasonably flexible. I have had the most success when using brass rods or steel wire, which you can find at art supply or architectural modelling stores. Learn from my mistakes and buy a sample rod of your chosen material first – that way you can check that you can make the shapes you want before you spend time attaching 30 gangly and uncooperative limbs!

The Trill sensor board has 30 pins, so you can add up to that number of limbs. For my harp creature, I used all 30 pins for limbs because I wanted a feathery, messy feel to match the generative nature of the code I planned to write, but for my bass creature I wanted a more minimal instrument with sharp, clean lines, so I only used ten pins – eight pins to span a full octave, plus two pins for triggering sound effects. Sketch out a few ideas, keeping in mind how the look might relate to the sound you want your own instrument to make.

In this make I'm using a Bela Mini to read the sensor values from the Trill Craft sensor and produce sound. However, the Trill Craft sensor board also works with Arduino, Teensy, Raspberry Pi, and any other microcontroller or single board computer that uses I2C communication. I chose the Bela Mini because I wanted low latency plus the ability to use Pure Data to produce a rich, complex sound for my creature, but you should feel free to make a decision that suits you.



> **"The most fragile part of this make is where your limbs attach to your sensor board with solder"**

Once you're happy with your design decisions, you can start soldering the limbs. Insert one of your metal rods into your sensor board and solder into place. When soldering the limbs, you'll need to apply your soldering iron for a little longer than you're used to with a normal component. This is because the limbs are long, thick, and made of metal, so they absorb and disperse the heat from your iron more than the little leg of a resistor would.

## STEP TWO SECURE SOLDER JOINTS

The most fragile part of this make is where your limbs attach to your sensor board with solder, especially if you've chosen limbs that are long and wiggly. The movement of the limbs backwards and forwards can eventually break the connection, leaving you with a useless limb and possibly even a broken pin connection.

With this in mind, it's worth pausing after you've soldered the first couple of limbs to your sensor board to think about how you can make them more secure. There are a couple of ways to do this, depending on how much wear and tear you expect your creature to endure. The simplest solution is to add a thin layer of hot glue at the end of soldering your limbs. This will give you a little extra support and insulation for your solder joints. If you used flux to help you solder, make sure to clean any excess off before sealing it up with hot glue.

The second method I experimented with to secure my limbs was a bit more effort but also a lot more protective and, to my eyes, much more aesthetically →

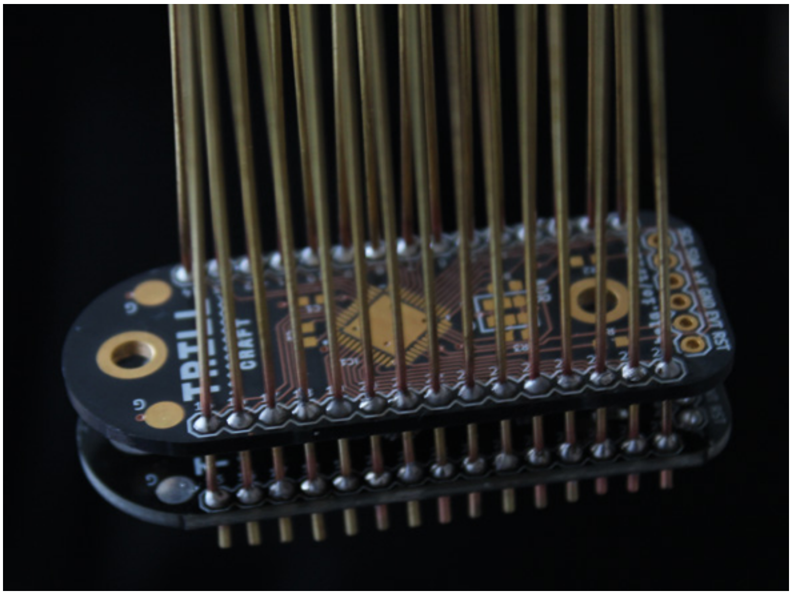
## YOU'LL NEED

### Tools

- **Soldering iron**
- **Helping hands**
- **Laptop or desktop computer**

### Materials

- **Trill Craft sensor**
- **Trill Craft bare PCB** (optional, see step three)
- **Bela Mini**
- **Four M-to-F jumper cables**
- **5 V power cable with ground**
- **0.8 mm thick metal rod or wire**



pleasing. To reduce the mechanical stress on the sensor board itself, I made a sensor board ‘sandwich’, with a functional board underneath and a dummy board on top to take the stress of the movement of the limbs. Because the lovely people at Bela have made their hardware open-source, I downloaded the KiCad PCB design file for the Trill Craft sensor board from their GitHub page (**hsmag.cc/BelaDesign**), then sent that file to OSH Park and placed an order for three boards. I used OSH Park because I wanted their gold and black ‘After Dark’ finish to match the rest of my creature, but you can use any PCB service for hobbyists. A week or so later I got a set of sensor boards in the post. Of course, these sensor boards didn’t have any components on them, but that didn’t matter to me – all I wanted was a nice-looking board to relieve the strain of my wiggly limbs.

I used this bare board to protect my Trill Craft board. If you choose to go this route too, then once you’ve soldered the first couple of limbs to your working sensor board, you can carefully thread the strain relief board onto those same limbs. Push it down until it’s nearly on top of your functioning Trill Craft, then carefully solder it in place. Once you have secured both of your boards in place, you then can move on to soldering the rest of your limbs.

**Above** An extra PCB helps hold the wires in place

**Below** Take your time with the soldering and make sure you get a good connection

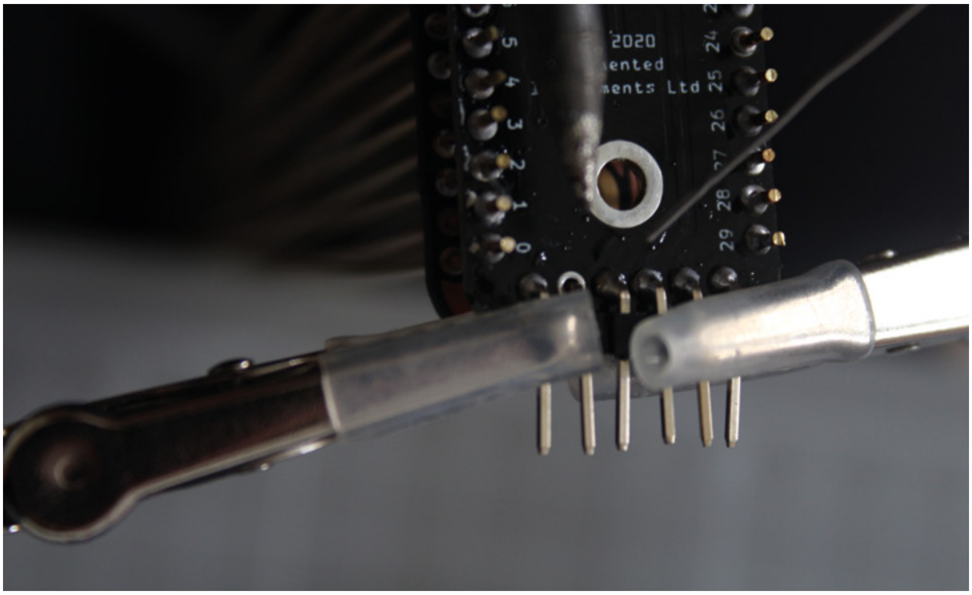
### STEP FOUR ADDING HEADER PINS

The Trill sensor comes with a row of header pins, so after the limbs are in place you need to solder them onto your board. I chose to solder the header pins underneath the sensor board, but if you prefer the way the pins look the way other around, do feel free to switch it up – as long as you wire it up to your Bela correctly it doesn’t matter. I soldered on all six header pins, but we’re only going to be making use of four for this make: SCL, SDA, +V, and GND.

### STEP FIVE SUPPORTING AND SHAPING YOUR CREATURE’S LIMBS

Once you have soldered and secured your limbs and added in your header pins, you can start to shape your creature. Depending on your design, you may want to add in support or a base at this point. The legs of my bass creature were sturdy and straight so they supported its body, but for this delicate spidery creature I used two M3 brass standoffs, two M3 nuts, and a bit of scrap wood to make a basic support stand. I secured the Trill Craft to one of the brass standoffs using an M3 bolt through the hole at the back of the board, then added a second brass standoff to give it a little extra height. Finally, I drilled a 3 mm hole in some scrap wood to allow space for the second M3 bolt to go through and connect to the brass standoffs.

Once you’re confident in the stability of your creature, gently and carefully bend each of the limbs of your creature into place. You can also trim the limbs at this point. This stage is where you really create the look of your creature, so think about where it will live, how you will play it, and what kind of aesthetic you want to achieve.



Whatever look you decide to go for, make sure that the limbs are not in contact with each other, or you will end up unintentionally triggering pins too often when you play it.

## STEP SIX WIRING YOUR CREATURE

Once you are happy with your creature's appearance, use the header pins and four jumper cables to wire it to your Bela Mini. Wire +V on the sensor board to VOUT, GND to GND, SCL to SCL, and SDA to SDA. If you flip the Bela Mini over, you will find the pins labelled on the bottom, or you can open up the IDE (see below) to find an interactive pin diagram. Once your creature is wired to your board, you need to connect it to your computer using a micro USB cable.

If you choose to use a different microcontroller or single-board computer, the wiring will be the same, but the instructions for setting up your board and making sounds will differ. For Linux-based systems including the Raspberry Pi, head over to Bela's GitHub for libraries and examples (**hsmag.cc/BelaPlatform**). You also can find libraries and examples for Arduino boards and Arduino-compatible systems, including Teensy, in a different folder on the Bela GitHub (**hsmag.cc/BelaArduino**).

## STEP SEVEN PREPARING YOUR AUDIO SOFTWARE

For my creature, I worked with the sound artist and composer Andrew Hockey to make a delicate, ethereal harp sound using Pure Data. Pure Data (often called Pd) is an open-source visual programming environment for sound, visuals, and other media. There are two versions of this software: Pd Vanilla and Purr Data. Either will work for this project, but I'm using Pd Vanilla. Go to **puredata.info**, then download and install the software onto your laptop or desktop computer.

Please note that Pure Data files will run on Bela and Raspberry Pi boards but not on a microcontroller. If you are using your Trill Craft sensor board with a microcontroller, you will have to make your own sounds. There will be many excellent audio libraries out there to play with, no matter which system you choose to use.

> **"There will be many excellent audio libraries out there to play with,** no matter which system you choose to use"

## STEP EIGHT WORKING WITH THE BELA IDE

One of the nice things about Bela is that you access the IDE using your web browser. This means you can connect your Bela to your computer, open a browser, and get coding – no downloads or special software needed! You don't even need to be connected to the internet: once your Bela is plugged in, your computer recognises it as a local network.

Plug your Bela system into your computer with a USB cable. After it's booted, go to bela.local in a web browser to load the Bela IDE. If bela.local does not bring up the IDE, try the IP addresses 192.168.6.2 or 192.168.7.2.

There are four main parts of the IDE: the editor, the toolbar, the tabs, and the console.

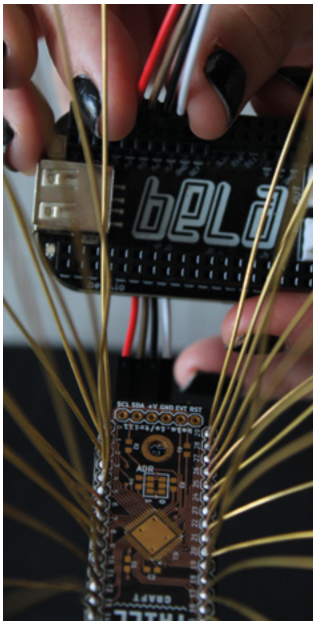
The editor is where you type your code: C++, Csound or SuperCollider programming language (sclang), or JavaScript if you're making a GUI. For this project we're using a Pure Data file, known as a patch. You can't edit a Pure Data patch in your browser, but once you've uploaded your patch it will be visualised in the editor.

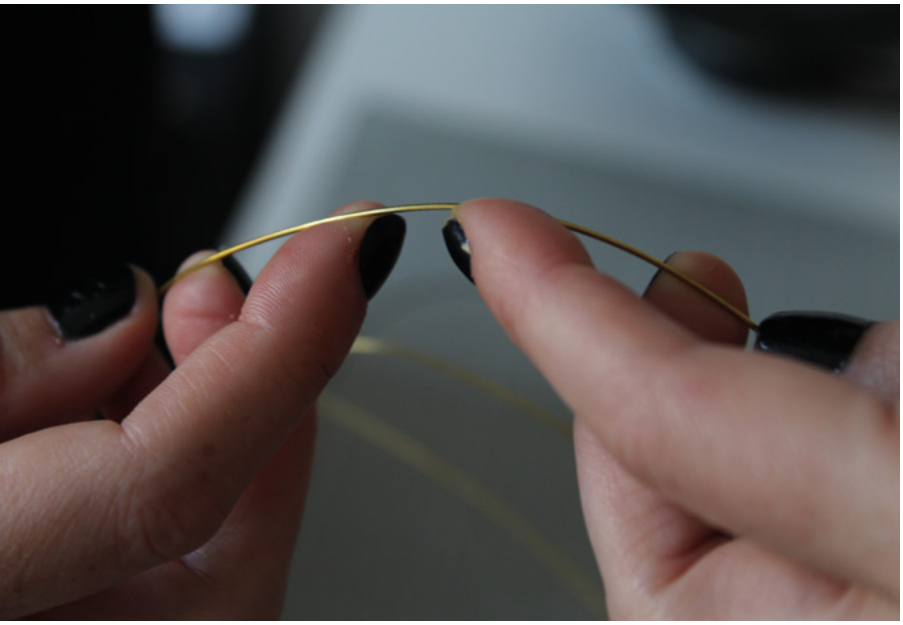
The toolbar contains controls for running and stopping your project as well as two useful tools: the oscilloscope (super-handy for visualising and examining signals) and the GUI visualiser.

The tabs contain most of the controls for your Bela system. This is where you'll manage your projects and customise settings, plus you can find lots of helpful →

**Above**
Shaping the instrument gives it its unique aesthetic

**Below**
Four wires join the two boards together





**Above** Brass wire is easy to bend and holds its shape well

resources here including code examples, an interactive pin diagram, and all the built-in code libraries.

The console gives you printed feedback about what your Bela is doing. You can print sensor data to this console as your code runs, and it will also provide useful error and status information. The console is also the command line for your Bela, letting you run terminal commands.

Go ahead and poke around the Bela IDE to get used to where some of the controls live. You can also test everything is working by running Bela's equivalent of 'Hello world!' or 'Blink': the sine tone. Click on the little light bulb icon in the tab section of the IDE to bring up the Examples folder. Click on Fundamentals, then click on the example project called 'sinetone' to open it up in your editor. Click on the 'Build & run' circular arrow icon in your toolbar and your code should start running on the board.

To hear the sine tone, you'll need to connect the audio adapter cable supplied with the board to 'out', then plug in your headphones or a speaker. If you hear a continuous bloop, you've successfully made your Bela make its first noise! Hooray! Now let's make it do something that sounds a little nicer.

## STEP NINE TESTING OUT YOUR CREATURE

Go back to the Examples tab, then scroll down until you see a folder called Trill. Click on that, then click on the project called 'craft-pd' to open it in the editor. You'll be able to see what the example Pure Data patch looks like, but you won't be able to edit it. Make sure your creature is connected to your Bela correctly and that none of the limbs are touching each other, either directly or through a conductive surface, then build and run the code using the circular arrow icon in your toolbar.

If everything is working correctly, your console should start printing 'bela: touchSensor: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0'. If you get an error message that says 'Unable to create Trill CRAFT device 'mycraft' on bus 1 at default address. Is the device connected?', that means something has gone wrong with your wiring. Double-check your SDA, SCL, +V, and GND wires are going where they should and try again.

Once your sensor has connected correctly, you can try touching some of the limbs on your creature. Some of the zeroes printed in your console should change and start returning different values. Each of these values corresponds to a different pin on your board. If you touch a limb attached to pin 0, 1, 2, or 3, you should be able to hear someone speaking.

## STEP NINE CHANGING THE DEMO CODE

We can't edit Pd files directly in the browser, so we need to download the example code to our computer. Click on the folder icon in the tabs section of the IDE to open the Project Explorer, then download your project and unzip the folder. Open the file **_main.pd** using the Pure Data software you installed earlier.

> "We can't edit Pd files directly in the browser, so we need to download the example code to our computer"

Once your file is open, go to the Edit menu and turn on 'Edit mode'. For our first edit we are going to make any pin we choose responsive, not just the four pins in the example code. Each of the pins can be accessed by editing the connections coming from the 'unpack f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f' block. Each of the black rectangles at the bottom of this block corresponds to a pin on the Trill Craft board. The first four rectangles have lines connecting them to a group of six blocks which trigger audio files. Select, copy, and paste one of these groups, then move this new group to its own space.

Hover your cursor over the first available unconnected black rectangle on the 'unpack' block, then click and drag the line to the top-left black rectangle of your new group. This can be a little fiddly with Pd Vanilla as you have to let go in just the right spot – watch for the circle to tell you you're in the right place! To give yourself more room to manoeuvre, you can increase the size of the 'unpack' block by dragging the right-hand edge. Next, connect the 'playAudioFile~' block to both the left and the right side of the 'dac~ 1 2'. Finally, change the name of the audio file in your new group of blocks to 'five.wav'.

Repeat this process for as many pins as you like, changing the audio file name for each pin, then save your patch. Your patch must be called **_main.pd** or your Bela will not be able to read it.

Once you're done with your patch, you can choose your files. Go to the project folder you downloaded and you will find four WAV files. To change the sounds triggered and add new ones, you need to save your sounds as WAV files in this folder with file names that match the file names you put in your Pure Data patch. For my edited patch, I triggered different miaows for each pin just to make myself laugh, but you can do whatever pleases you most.

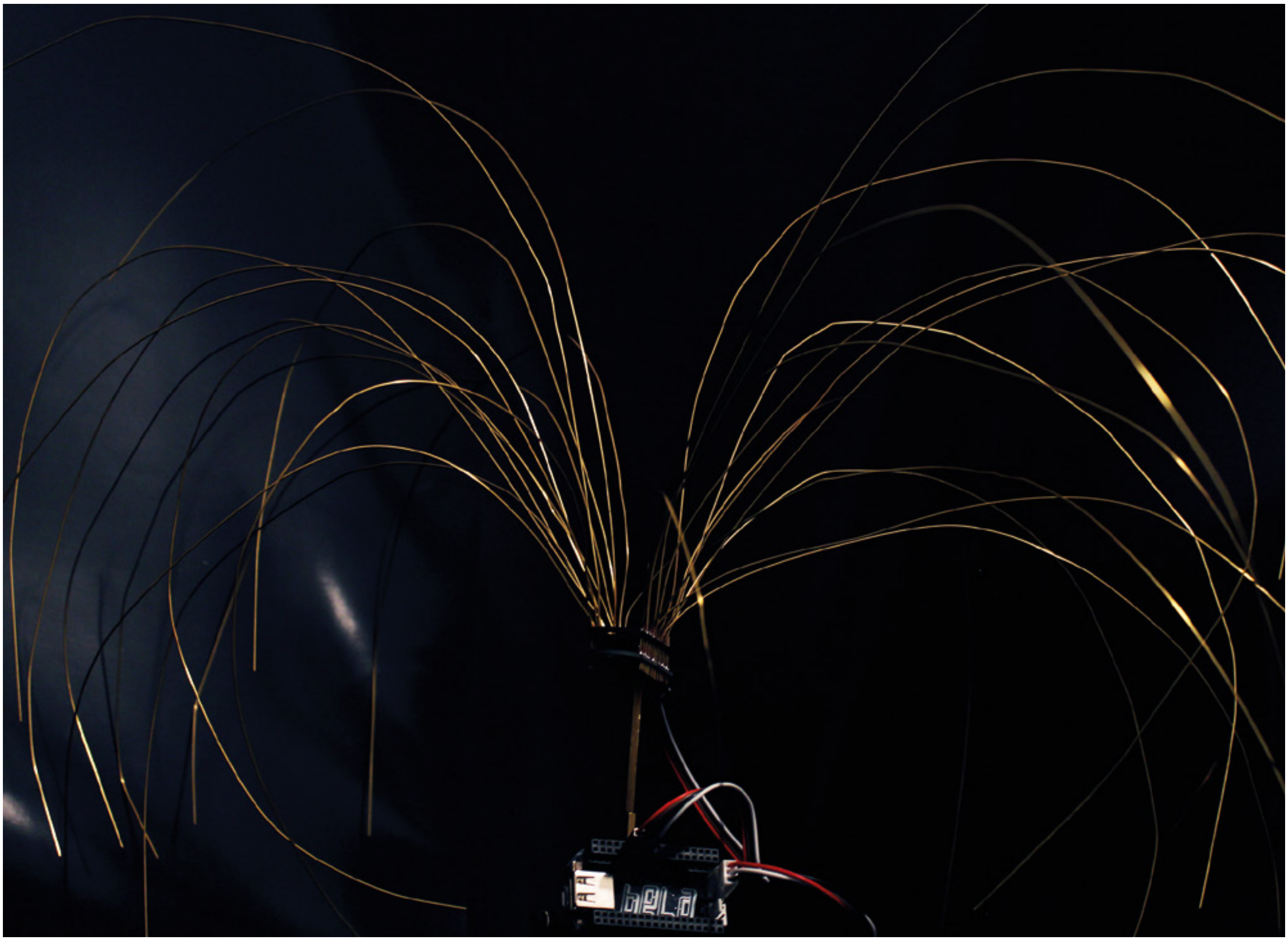
### STEP TEN UPLOADING YOUR NEW PATCH

Once you are happy with your edited patch and new sounds, you need to upload it all to your board.

Go back to the IDE in your browser and create a new Pure Data project. Go to the Project Explorer tab and click the Create New Project button at the top. Select Pure Data in the dialog box that pops up, give your project a name, and click 'Create project' to make a blank Pure Data Bela project. Click on Project Contents in the Project Explorer tab and you will see that the only thing in it is a file called **_main.pd**. You can either upload your files individually in the IDE, or you can drag all the files from the folder on your computer and drop them anywhere on the Bela IDE window. This will upload and run your patch, ready for you to test out your new samples and additional pins.

Once you have a Pd file you're happy with, you can disconnect your Bela from your computer and power the creature using a 5V power cable. ☐

**Below**
This is my instrument, but it's up to you how you build yours



# NeoPixel dithering with Pico

Get extra levels of brightness out of your LEDs

**Ben Everard**

@ben_everard

Ben's house is slowly being taken over by 3D printers. He plans to solve this by printing an extension, once he gets enough printers.

**WS2812B LEDs, commonly known as NeoPixels, are cheap and widely available LEDs.** They have red, green, and blue LEDs in a single package with a microcontroller that lets you control a whole string of them using just one pin on your microcontroller. However, they do have a couple of disadvantages:

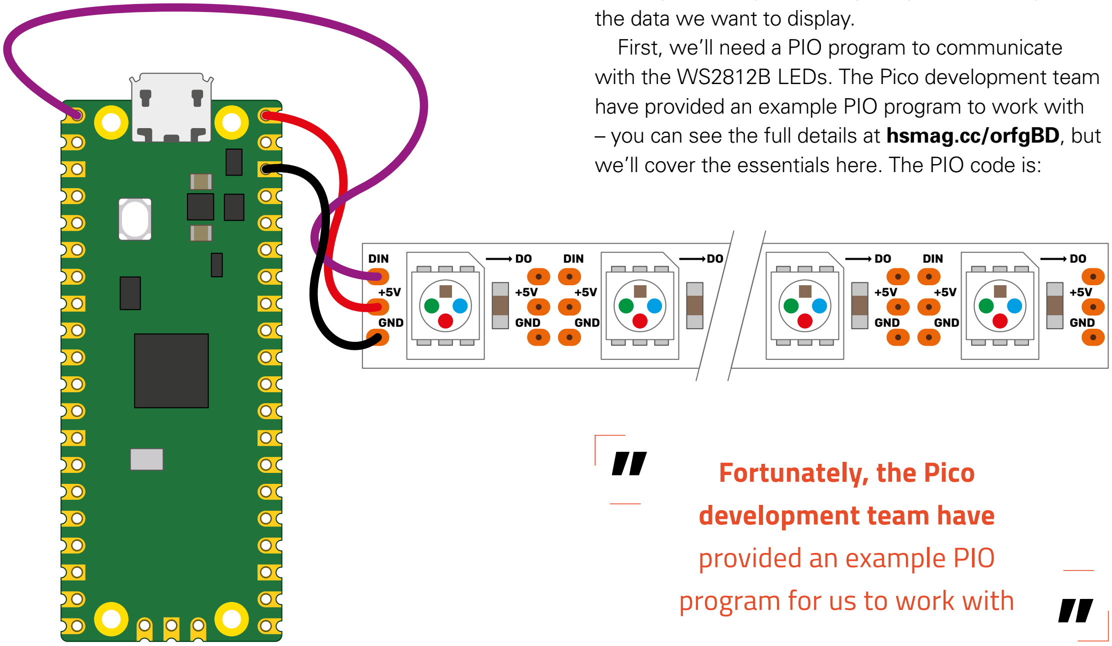
1) The protocol needed to control them is timing-dependent, and often has to be bit-banged.

2) Each colour has 8 bits, so has 255 levels of brightness. However, these aren't gamma-corrected, so the low levels of brightness have large steps between them. For small projects, we often find ourselves only using the lower levels of brightness, so often only have 10 or 20 usable levels of brightness.

We're going to look at how two features of Pico help solve these problems. Firstly, Programmable IO (PIO) lets us implement the control protocol on a state machine rather than the main processing cores. This means that we don't have to dedicate any processor time to sending the data out.

Secondly, having two cores means we can use one of the processing cores to dither the NeoPixels. This means shift them rapidly between different brightness levels to make pseudo-levels of brightness. For example, if we wanted a brightness level halfway between levels 3 and 4, we'd flick the brightness back and forth between 3 and 4. If we can do this fast enough, our eyes blur this into a single brightness level and we don't see the flicker. By varying the amount of time at levels 3 and 4, we can make many virtual levels of brightness. While one core is doing this, we still have a processing core completely free to manipulate the data we want to display.

First, we'll need a PIO program to communicate with the WS2812B LEDs. The Pico development team have provided an example PIO program to work with – you can see the full details at **hsmag.cc/orfgBD**, but we'll cover the essentials here. The PIO code is:



**Right** ↗
The three connections may be in a different order on your LED strip, so check the labels to make sure they're connected correctly

> **Fortunately, the Pico development team have** provided an example PIO program for us to work with

**Above** There will usually be wires already connected to your strip, but if you cut it, you'll need to solder new wires on

```
.program ws2812
.side_set 1
.define public T1 2
.define public T2 5
.define public T3 3

bitloop:
    out x, 1       side 0 [T3 - 1]
    jmp !x do_zero side 1 [T1 - 1]
    do_one:
    jmp  bitloop   side 1 [T2 - 1]
    do_zero:
    nop            side 0 [T2 - 1]
```

We looked at the PIO syntax in the main cover feature, but it's basically an assembly language for the PIO state machine. The WS2812B protocol uses pulses at a rate of 800kHz, but the length of the pulse determines if a 1 or a 0 is being sent. This code uses jumps to move through the loop to set the timings depending on whether the bit (stored in the register `x`) is 0 or 1. The `T1`, `T2`, and `T3` variables hold the timings, so are used to calculate the delays (with 1 taken off as the instruction itself takes one clock cycle).

There's also a section in the **pio** file that links the PIO code and the C code:

```
% c-sdk {
#include "hardware/clocks.h"

static inline void ws2812_program_init(PIO pio,
uint sm, uint offset, uint pin, float freq, bool
rgbw) {

    pio_gpio_select(pio, pin);
    pio_sm_set_consecutive_pindirs(pio, sm, pin, 1,
true);
    pio_sm_config c = ws2812_program_get_default_
```

→

## DIFFERENT LEDS

WS2812B LEDs come in a lot of different types, and many LEDs that are sold as WS2812N LEDs aren't really WS2812B LEDs at all. You may find that the colours don't match up properly with what we're getting here (the colour order can be different). You should be able to amend the code to fix this. If you don't get anything sensible out (or just odd lights), it may be that you have 400kHz LEDs rather than 800kHz ones. In which case, you'll need to decrease the frequency in the `ws2812_program_init` line in `int main`.

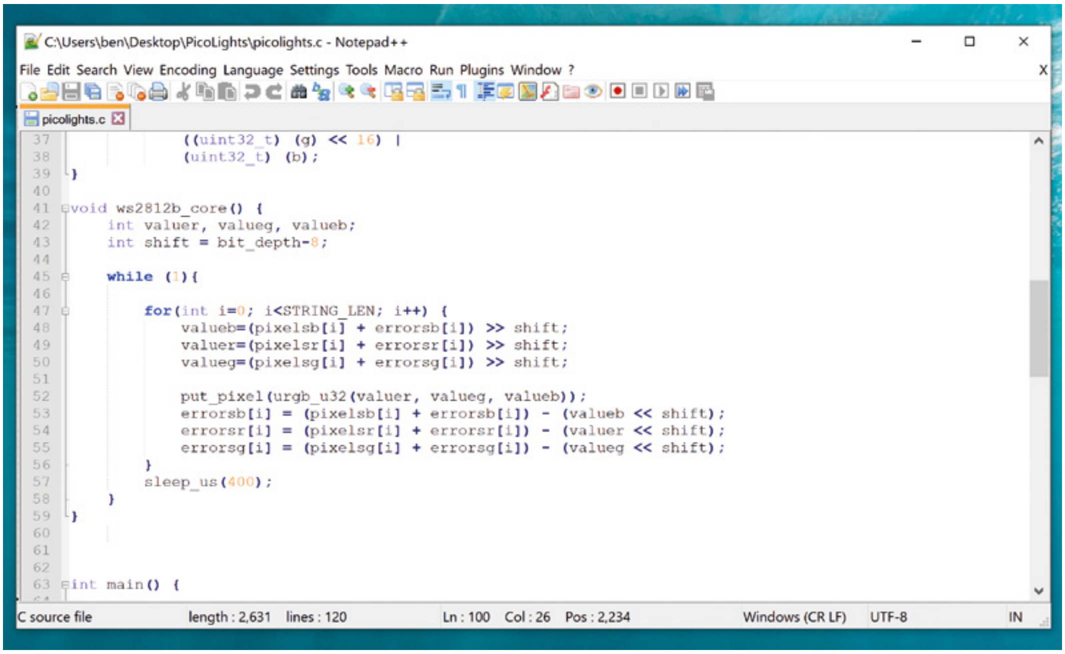
## SETTING UP THE SDK

In order to compile this code, you need to set up the SDK. Follow the steps at **pico.raspberrypi.org/getting-started**. Once you've got that set up, you can use CMake to build the project. So, if you have two directories at the same level, one called **PicoLights** and one called **PicoLights-build**, you can build the project with the following run in the **PicoLights-build** directory:

```
cmake ../PicoLights -G "Nmake Makefiles"
nmake
```

This is for Windows. You'll need to alter the `-G` parameter of CMake if you're using macOS or Linux. See the getting started documentation for details.

If you'd rather just see it running, you can get the compiled code from **github.com/benevpi/PicoLights/releases/tag/v.0.1** and flash this to your Pico.


C:\Users\ben\Desktop\PicoLights\picolights.c - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
picolights.c
C source file
length : 2,631 lines : 120
Ln : 100 Col : 26 Pos : 2,234
Windows (CR LF)
UTF-8
IN

```
                ((uint32_t) (g) << 16) |
                (uint32_t) (b);
}

void ws2812b_core() {
    int valuer, valueg, valueb;
    int shift = bit_depth-8;

    while (1){

        for(int i=0; i<STRING_LEN; i++) {
            valueb=(pixelsb[i] + errorsb[i]) >> shift;
            valuer=(pixelsr[i] + errorsr[i]) >> shift;
            valueg=(pixelsg[i] + errorsg[i]) >> shift;

            put_pixel(urgb_u32(valuer, valueg, valueb));
            errorsb[i] = (pixelsb[i] + errorsb[i]) - (valueb << shift);
            errorsr[i] = (pixelsr[i] + errorsr[i]) - (valuer << shift);
            errorsg[i] = (pixelsg[i] + errorsg[i]) - (valueg << shift);
        }
        sleep_us(400);
    }
}



int main() {
```

**Above** Using a text editor with programmer's features such as syntax highlighting will make the job a lot easier

```
config(offset);
    sm_config_set_sideset_pins(&c, pin);
    sm_config_set_out_shift(&c, false, true, rgbw ?
32 : 24);
    sm_config_set_fifo_join(&c, PIO_FIFO_JOIN_TX);

    int cycles_per_bit = ws2812_T1 + ws2812_T2 +
ws2812_T3;
    float div = clock_get_hz(clk_sys) / (freq *
cycles_per_bit);

    sm_config_set_clkdiv(&c, div);
    pio_sm_init(pio, sm, offset, &c);

    pio_sm_set_enable(pio, sm, true);
}
%}
```

Most of this is setting the various PIO options – the full range is detailed in the Pico C/C++ SDK document at **rptl.io/rp2040**.

```
 sm_config_set_out_shift(&c, false, true, rgbw ? 32
: 24);
```

This line sets up the output shift register which holds each 32 bits of data before it's moved bit by bit into the PIO state machine. The parameters are the config (that we're setting up and will use to initialise the state machine); a Boolean value for shifting right or left (false being left); and a Boolean value for autopull which we have set to true. This means that whenever the output shift register falls below a certain threshold (set in the next parameter), the PIO will automatically pull in the next 32 bits of data.

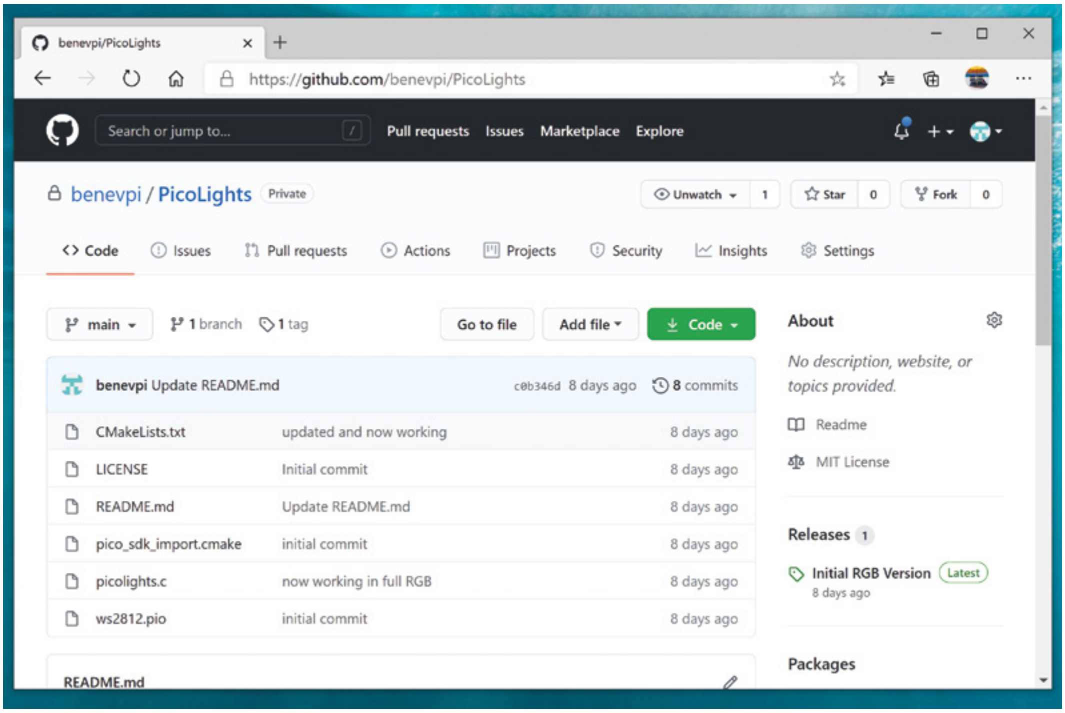
The final parameter is set using the expression `rgbw ? 32 : 24`. This means that if the variable `rgbw` is true, the value 32 is passed, otherwise 24 is passed. The `rbgw` variable is passed into this function when we create the PIO program from our C program and is used to specify whether we're using an LED strip with four LEDs in each using (one red, one green, one blue, and one white) or three (red, green, and blue).

The PIO hardware works on 32-bit words, so each

## SPEED

The protocol for WS2812B LEDs throws data out at 800kHz (well, there is a little wiggle room to speed this up). The total time it takes to update a string of LEDs obviously depends on how many LEDs you have on it. We found that this code worked well with about 60–80 LEDs. If you start adding more than this, then you may start to notice flicker.

This doesn't mean that you're limited to this many WS2812B LEDs in a total display, though, as you can add multiple state machines controlling multiple strings of LEDs and, as long as each string is kept short enough, you should have the processing power to manipulate a lot of LEDs. However, if you're going to do this, you probably want to use a few more techniques to squeeze out maximum performance. Take a look at the 'Going Further' box (overleaf) for more details.


benevpi/PicoLights
https://github.com/benevpi/PicoLights
Search or jump to...
Pull requests
Issues
Marketplace
Explore
benevpi / PicoLights
Private
Unwatch 1
Star 0
Fork 0
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Settings
main
1 branch
1 tag
Go to file
Add file
Code
About
Readme
MIT License
Releases 1
Initial RGB Version
Latest
8 days ago
Packages
README.md

benevpi Update README.md c0b346d 8 days ago 8 commits

| | | |
|---|---|---|
| CMakeLists.txt | updated and now working | 8 days ago |
| LICENSE | Initial commit | 8 days ago |
| README.md | Update README.md | 8 days ago |
| pico_sdk_import.cmake | initial commit | 8 days ago |
| picolights.c | now working in full RGB | 8 days ago |
| ws2812.pio | initial commit | 8 days ago |

*No description, website, or topics provided.*

**Left**
If you just want to see this in action, you can download the UF2 file from hsmag.cc/orfgBD and flash it straight to your Pico

chunk of data we write with the values we want to send to the LEDs has to be 32 bits long. However, if we're using RGB LED strips, we actually want to work in 24-bit lengths. By setting autopull to 24, we still pull in 32 bits each time, but once 24 bits have been read, another 32 bits are pulled in which overwrite the remaining 8 bits.

```
sm_config_set_fifo_join(&c, PIO_FIFO_JOIN_TX);
```

Each state machine has two four-word FIFOs attached to it. These can be used for one going in and one coming out. However, as we only have data going into our state machine, we can join them together to form a single eight-word FIFO using the above line. This gives us a small buffer of time to write data to in order to avoid the state machine running out of data and execution stalling.

The following three lines are used to set the speed the state machine runs at:

```
    int cycles_per_bit = ws2812_T1 + ws2812_T2 +
ws2812_T3;
    float div = clock_get_hz(clk_sys) / (freq *
cycles_per_bit);
    sm_config_clkdiv(&c, div);
```

The WS2812B protocol demands that data is sent out at a rate of 800kHz. However, each bit of data requires a number of state machine cycles. In this case, they're defined in the variables `T1`, `T2`, and `T3`. If you look back at the original PIO program, you'll see that these are used in the delays (always with 1 taken off the value because the initial instruction takes one cycle before the delay kicks in). Every loop of the PIO program will take `T1` + `T2` + `T3` cycles. We use these values to calculate the speed we want the state machine to run at, and from there we can work out the divider we need to slow the system clock down to the right speed for the state machine.

The final two lines just initialise and enable the state machine.

## THE MAIN PROCESSOR

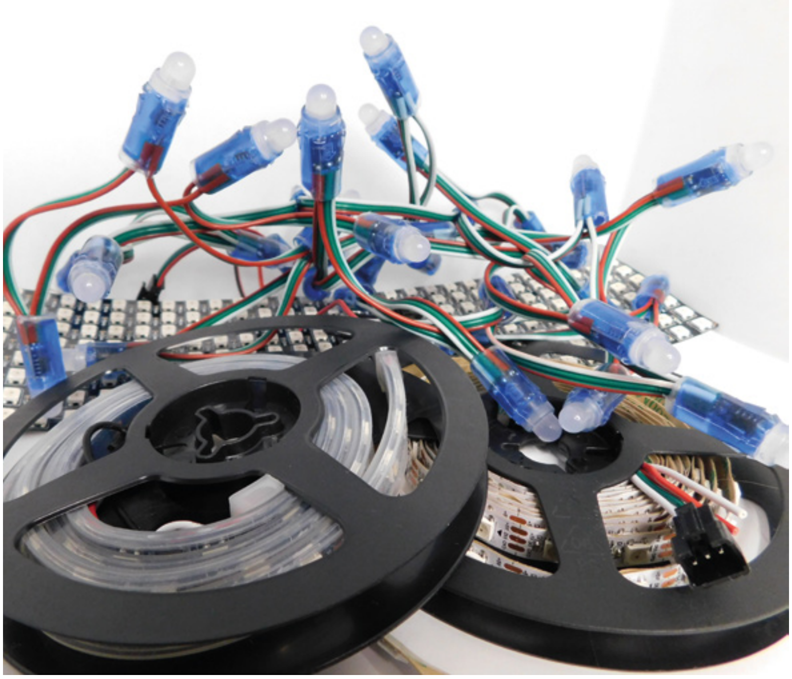
That's the code that's running on the state machine, so let's now look at the code that's running on our main processor cores.

The full code is at **github.com/benevpi/PicoLights**. Let's first look at the code running on the second core (we'll look at how to start this code running shortly), as this controls the light levels of the LEDs.

```
int bit_depth=12;
const int PIN_TX = 0;

uint pixels[STRING_LEN];
uint errors[STRING_LEN];
```

→



**Above** NeoPixels come in many different shapes and sizes

```
static inline void put_pixel(uint32_t pixel_grb) {
    pio_sm_put_blocking(pio0, 0, pixel_grb << 8u);
}

static inline uint32_t urgb_u32(uint8_t r, uint8_t
g, uint8_t b) {
    return
            ((uint32_t) (r) << 8) |
            ((uint32_t) (g) << 16) |
            (uint32_t) (b);
}

void ws2812b_core() {
   int valuer, valueg, valueb;
   int shift = bit_depth-8;

    while (1){

     for(int i=0; i<STRING_LEN; i++) {
       valueb=(pixelsb[i] + errorsb[i]) >> shift;
       valuer=(pixelsr[i] + errorsr[i]) >> shift;
       valueg=(pixelsg[i] + errorsg[i]) >> shift;
       put_pixel(urgb_u32(valuer, valueg, valueb));
       errorsb[i] = (pixelsb[i] + errorsb[i]) -
(valueb << shift);
       errorsr[i] = (pixelsr[i] + errorsr[i]) -
(valuer << shift);
       errorsg[i] = (pixelsg[i] + errorsg[i]) -
(valueg << shift);
     }
     sleep_us(400);
   }
}
```

We start by defining a virtual bit depth. This is how many bits per pixel you can use. Our code will then attempt to create the necessary additional brightness levels. It will run as fast as it can drive the LED strip, but if you try to do too many brightness levels, you'll start to notice flickering. We found twelve to be about the best with strings up to around 100 LEDs, but you can experiment with others.

Our code works with two arrays – `pixels` which holds the values that we want to display, and `errors` which holds the error in what we've displayed so far (there are three of each for the different colour channels). To explain that latter point, let's take a look at the algorithm for determining how to light the LED. We borrowed this from the source code of Fadecandy by Micah Scott, but it's a well-used algorithm for calculating error rates.

We have an outer `while` loop that just keeps pushing out data to the LEDs as fast as possible. We don't care about precise timings and just want as much speed as possible. We then go through each pixel. The corresponding item in the `errors` array hold the cumulative amount our LED has been underlit so far compared to what we want it to be. Initially, this will be zero, but with each loop (if there's a difference between what we want to light the LED and what we can light the LED) this error value will increase.

These two numbers (the closest light level and the error) added together give the brightness at the pseudo-level, so we need to bit-shift this by the difference between our virtual level and the 8-bit brightness levels that are available. This gives us the value for this pixel which we write out. We then need to calculate the new error level.

Let's take a look at what this means in practice. Suppose we want a brightness level halfway between 1 and 2 in the 8-bit levels. To simplify things, we'll use nine virtual bits. 1 and 2 in 8-bit is 2 and 4 in 9 bits (adding an extra 0 to the end multiplies everything by a power of 2), so halfway between these two is a 9-bit value of 3 (or 11 in binary, which we'll use from now on).

## GOING FURTHER

There are quite a few ways of improving this code. Firstly, you can use DMA to give you more headroom for timings. Secondly, a single PIO can drive more than one output, and therefore more than one string of WS2812B LEDs. However, these changes do increase the complexity of the code.

You can take a look at the **ws2812b_parrallel.c** example for details of how to implement these.

In the first iteration of our loop, **`pixels`** is 11, **`errors`** is 0, and **`shift`** is 1.

```
value = 11 >> 1 = 1
errors = 11 - 10 = 1
```

So this time, the brightness level of 1 is written out. The second iteration, we have:

```
value = 100 >> 1 = 10
errors = 100 - 100 = 0
```

So this time, the brightness level of 10 (in binary, or 2 in base 10) is written out. This time, the errors go back to 0, so we're in the same position as at the start of the first loop. In this case, the LED will flick between the two brightness levels each loop so you'll have a brightness half way between the two.

Using this simple algorithm, we can experiment with different virtual bit-depths. The algorithm will always handle the calculations for us, but we just have to see what creates the most pleasing visual effect for the eye. The larger the virtual bit depth, the more potential iterations you have to go through before the error accumulates enough to create a correction, so the more likely you are to see flicker.

The biggest blocker to increasing the virtual bit depth is the **`sleep_us(400)`**. This is needed to reset the LED strip. Essentially, we throw out bits at 800kHz, and each block of 24 bits is sent, in turn, to the next LED. However, once there's a long enough pause, everything resets and it goes back to the first LED. How big that pause is can vary. The truth is that a huge proportion of WS2812B LEDs are clones rather than official parts – and even for official parts, the length of the pause needed to reset has changed over the years. 400 microseconds is conservative and should work, but you may be able to get away with less (possibly even as low as 50 microseconds for some LEDs).

The **`urgb_u32`** method simply amalgamates the red, blue, and green values into a single 32-bit string (well, a 24-bit string that's held inside a 32-bit string), and **`put_pixel`** sends this to the state machine. The bit shift there is to make sure the data is in the right place so the state machine reads the correct 24 bits from the output shift register.

## POWER

You can connect the power line in your WS2812B strip directly to the 5V output on your Pico, and this should work for a small strip of LEDs. We say 'should' because it's technically out-of-spec. The datasheet says that you need a data voltage of at least 0.8 times the power voltage. With 5V power, that means you need a data voltage of 3.5V. In practice, however, almost all the WS2812B LEDs we've tested work absolutely fine at 3.3V, which is what Pico's GPIOs use.

Another power problem is that the LEDs can create a lot of noise on the power line. To get different brightness levels, the LEDs flick on and off, and this can create problems as they demand more or less electricity. Adding a smoothing capacitor between the 5V and GND will help eliminate any problems. If you're still having problems after you've added a capacitor, it could be the voltage level. You can either use a logic level shifter to increase the voltage on the data line, or add a 0.7V voltage drop diode to the voltage line.

## GETTING IT RUNNING

We've now dealt with all the mechanics of the code. The only bit left is to stitch it all together.

```
int main() {

   PIO pio = pio0;
   int sm = 0;
   uint offset = pio_add_program(pio, &ws2812_
program);
    ws2812_program_init(pio, sm, offset, PIN_TX,
1000000, false);
    multicore_launch_core1(ws2812b_core);

    while (1) {
        for (int i = 0; i < 30; ++i) {
         pixels[i] = i;

         for (int j=0;j<30;++j){
           pixels[0] = j;
             if(j%8 == 0) { pixels[1] = j; }
               sleep_ms(50);
             }
          for (int j=30;j>0;--j){
           pixels[0] = j;
              if(j%8 == 0) { pixels[1] = j; }
             sleep_ms(50);
           }
        }
    }
}
```

The method **`ws2812_program_init`** calls the method created in the PIO program to set everything up.

To launch the algorithm creating the virtual bit-depth, we just have to use **`multicore_launch_core1`** to set a function running on the other core. Once that's done, whatever we put in the **`pixels`** array will be reflected as accurately as possible in the WS2812B LEDs. In this case, we simply fade it in and out, but you could do any animation you like.

# Touchscreen laser control

Build a Raspberry Pi-powered control system for your K40 laser

**Dr Andrew Lewis**

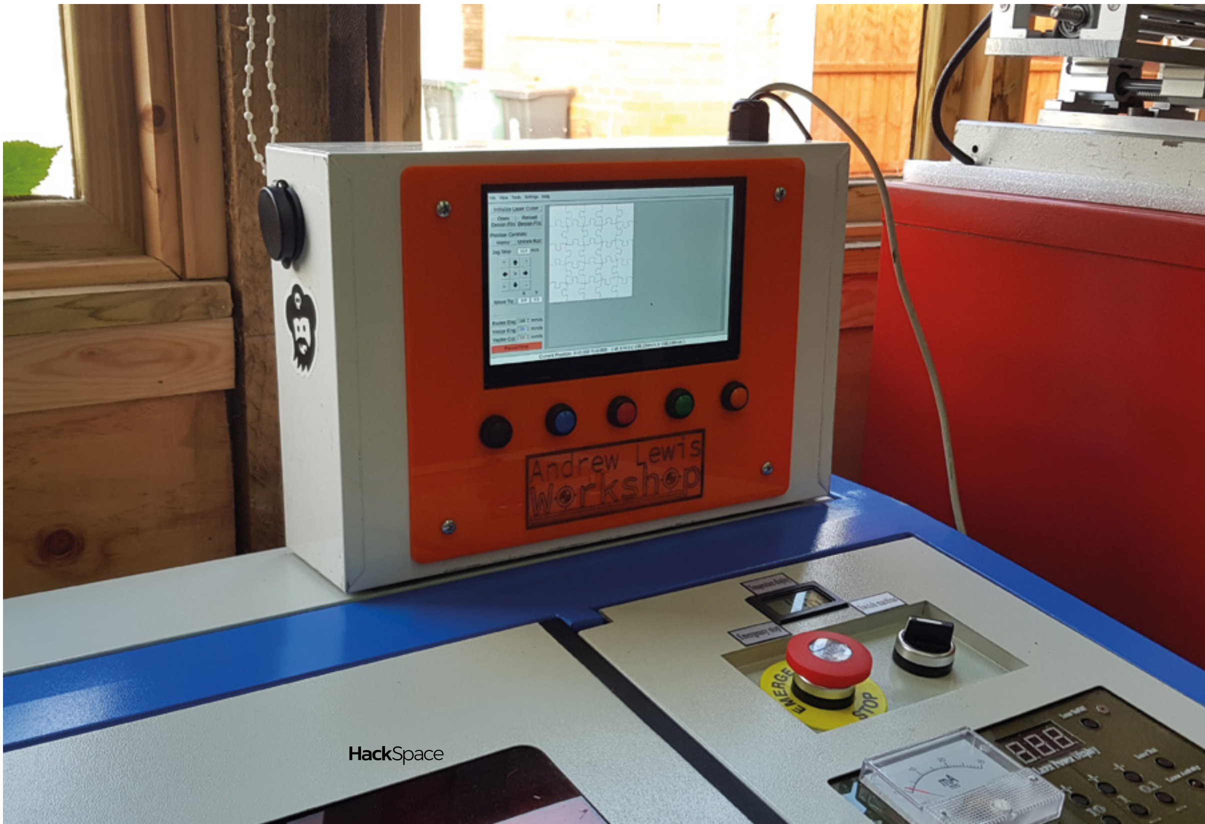
Dr Andrew Lewis is a specialist fabricator and maker, and is the owner of the Andrew Lewis Workshop.

**A K40 laser cutter is a nice addition to a workshop, but it takes a lot of work to get the best out of it.** Aside from the obvious safety and performance upgrades, one of the best things you can do for your K40 is to add dedicated computer control. In this article, you will learn how to add a Raspberry Pi to connect your K40 to your WiFi network, with a touchscreen and configurable push-button controls.

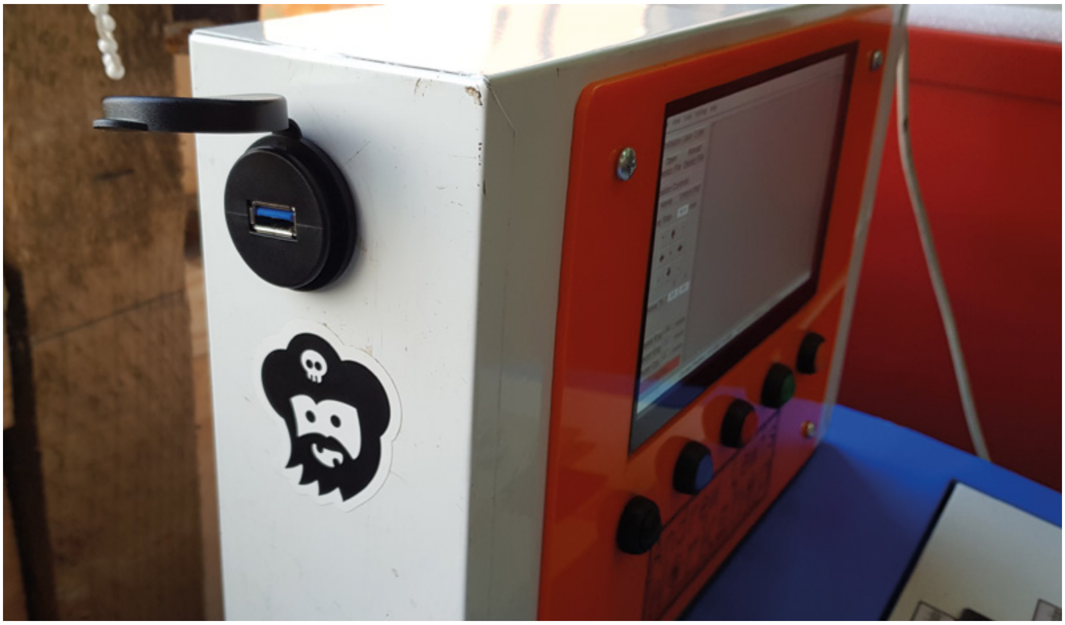
To make this project, you will be wiring the switches and touchscreen to the Raspberry Pi, and housing it inside the metal cabinet. To make the mounting process easier, you can use the bezel template for this project to cut the screen mount on your laser cutter. Mount the push-button switches into the bezel, and then line up the touchscreen with the cut-out in the bezel. The screen should push into the cut-out and be a tight fit. It might be necessary to shave a little bit of the plastic bezel away with a razor blade, depending on the kerf of your laser cutter. Use hot glue around the back of the screen to hold it in place. Cut a hole in the back of the medical cabinet, about 19 cm × 15 cm, and mount the bezel over this hole so that the screen and buttons are inside the hole.

## WIRELESS LASER FUN

Transfer-mark the holes in the bezel to the metal cabinet, and drill them out. Next, cut out a hole

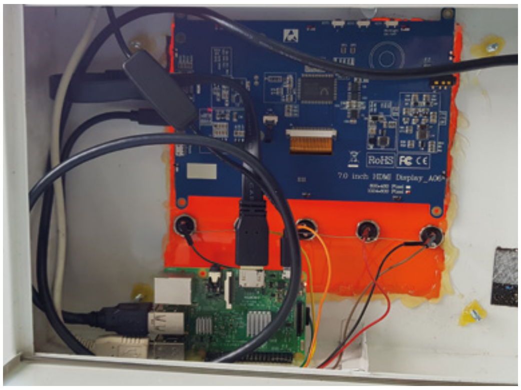


**Right**
Keep the control of your K40 laser cutter local by adding a Raspberry Pi. Shorter USB cables mean less chance of CRC errors and accidental cable disconnections



in the side of the cabinet for the panel-mounted USB socket. The position of this socket is down to personal preference, and can be placed on either side or even on the top of the cabinet. Place another hole in the case to pass the USB power for Raspberry Pi, and the USB cable for the laser. You can make this hole look neater by using a 20 mm cable gland, which will also help to protect the cables from the sharp edges of the metal cabinet. Fix the bezel in place on the cabinet with the bolts, and improve the fit by running hot glue along the internal edges of the cut-out in the metal box so that the box is hot-glued to the plastic bezel all the way around the inside.

Mount your Raspberry Pi into the medicine cabinet, and connect it to the touchscreen using the provided USB and HDMI cables. The push-buttons need to be connected next, by soldering one side of each button to a ground pin on Raspberry Pi, and the other side of the buttons to GPIO pins 26, 19, 13, 6, and 5. By default, these buttons will be configured →



## UNTOUCHABLE

The default version of K40 Whisperer doesn't work as we need it to for this project. It doesn't work well with a touchscreen unless you have a keyboard attached, doesn't start up in full-screen mode, and it doesn't connect to the push-buttons on Raspberry Pi. You'll need to make some modifications to the code to get this working. Bind the push-buttons to commands by adding the following code around line 200 of **k40_whisperer.py**:

```
self.master.bind('<q>' , self.Raster_Eng)
self.master.bind('<w>' , self.Vector_Eng)
self.master.bind('<e>' , self.Vector_Cut)
self.master.bind('<r>' , self.Home)self.
master.bind('<t>' , self.Unlock)
```

Make the app start full-screen by adding `root.attributes("-fullscreen", True)` at line 5734, just after the line reading `root= Tk()`. Next, you'll need to replace all of the text entry boxes with spin boxes so that their values can be changed without typing in numbers. As an example, line 473 should change from

```
self.Entry_Reng_feed = Entry(self.master,
width="15") to
self.Entry_Reng_feed = Spinbox(self.master,
width="15",values=(100,150,200,250,300),
state="readonly")
```

This change will make the text entry into a spinbox with fixed values, and disable the ability to manually type a value in the box. If this is starting to sound complicated, don't panic. A modified version of the code is available at **hsmag.cc/issue39**, and the majority of these changes are already done. Simply download the file and replace the existing **k40_whisperer.py** with the downloaded version.

## YOU'LL NEED

- **Raspberry Pi 3 or 4**
- **5 V 3 A PSU for Raspberry Pi**
- **7" touchscreen** (hsmag.cc/TouchScr)
- **USB female-to-male panel mount connector** (hsmag.cc/MntCon)
- **5 × 12 mm push-buttons momentary normally open** (hsmag.cc/PButton)
- **Metal medicine cabinet, 9 cm × 22 cm × 32.5 cm or similar** (hsmag.cc/Cabinet)
- **1 m USB A-B cable to connect Raspberry Pi to laser**
- **Plastic bezel – laser-cut from file**
- **Neodymium magnets to connect cabinet to laser chassis** (optional)
- **M20 cable gland**

**Above**
Make sure that you add a cover for your USB socket. Workshops are often dusty and dirty places, and it's better to keep the USB socket clear of any dust and debris

**Left**
You can mount your Raspberry Pi using board supports or a plastic case with a strip of Velcro, although hot-gluing the edge will work as a temporary solution

### QUICK TIP

Remember to change the default password to improve security on your Raspberry Pi OS install.

**Above** Fix your controller onto the laser cutter by adding a few magnets to the base of the cabinet. Old hard drive magnets are great for this, providing enough strength to hold the cabinet in place, while still allowing it to be easily detached when necessary

to start raster engraving, vector engraving, vector cutting, send the carriage to the home position, and turn off the gantry motors. There is no reason why you can't add more buttons or configure them to perform different tasks, and this will be explained in more detail further on in this article.

Install the latest version of Raspberry Pi OS on your Raspberry Pi, and check for updates. To set up Raspberry Pi as a laser controller, you will need to install the touchscreen drivers, install the Retrogame utility and configure it to work with the buttons in the bezel, install K40 Whisperer so that it runs automatically on startup, and modify the code so that it understands the button input and runs in full-screen when it's started.

The touchscreen may work straight out of the box – if that isn't the case, touchscreen installation is not difficult. You can follow the instructions at **hsmag.cc/DisplayC** to get going.

Adafruit provides an installer script for Retrogame. You can follow the instructions at **hsmag.cc/Retrogame** to install it. When you run the installer, you will be asked to choose a control configuration. It doesn't really matter which option you pick, because you will be creating your own configuration. Once the installer has finished, reboot your Raspberry Pi, and then open up **/boot/retrogame.cfg** in your text editor of choice. The Retrogame configuration file is very simple, consisting of a key name (like **A** or **SPACE**) and then a number that refers to the GPIO pin that this should be connected to. You are going to use Retrogame to connect the push-buttons to the **QWERT** keys, and then, later on, you will modify K40 Whisperer to monitor these keys and trigger events within the program. With Retrogame's simple config format, the entire config file should look like this:

```
Q 26
W 19
E 13
R 6
T 5
```

## AUTOSTART

Ideally, K40 Whisperer needs to start up automatically when you boot the Raspberry Pi. There are a number of methods available to accomplish this, but the easiest is probably to use autostart. Open a terminal and type the following:

```
mkdir /home/pi/.config/autostart
nano /home/pi/.config/autostart/k40.desktop
```

This will create an **autostart** directory and a new file called **k40.desktop**. In nano, enter the following lines:

```
[Desktop Entry]
Type=Application
Name=k40
Exec=python /home/pi/K40_Whisperer-0.52_
src/k40_whisperer.py
```

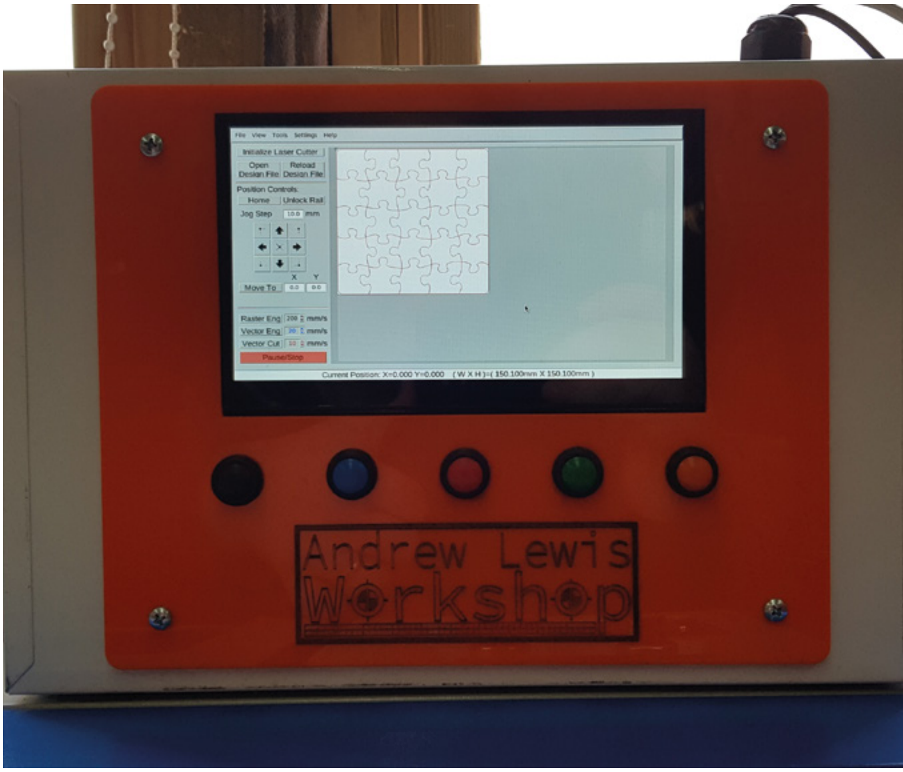
Save the file and reboot. K40 Whisperer should now start automatically, and you're almost done with the setup of the laser controller.

## PUSH FOR ACTION

If you are adding extra buttons, you just need to add them here, binding keys to GPIO pins as necessary. Restart the Raspberry Pi, and open a text editor. Pressing a push-button should now type a letter in the text editor. After confirming the buttons are working, you can move on to download the K40 Whisperer source from **hsmag.cc/Whisperer**. Follow the instructions in the **README_linux.txt** to install the necessary dependencies for K40 Whisperer, and then make sure that it runs by navigating to the source directory and typing `python k40_whisperer.py`.

## OUTSOURCE YOUR PROCESSING

Depending on the complexity of the file you are intending to laser-cut or etch, you might find that your Raspberry Pi takes a while to process design files. As a workaround to this, you can save time by processing the file on your desktop or laptop machine, and saving it as an EGV file to Raspberry Pi. If you choose to do this, it is probably worth changing the 'Reload design file' button in the interface to open EGV files instead. This has already been done in the modified **k40_whisperer.py** file provided with this project.



**Above** The buttons on the panel are colour-coded to match the SVG line colours used by K40 Whisperer. Black for raster engraving, blue for vector engraving, and red for vector cutting. The green and orange buttons are used to home the gantry and turn off the stepper motors

## SHARED ACCESS

While it's possible to copy files to the laser controller using a USB drive, it's easier to connect to the laser using a shared directory. Set up Samba on your Raspberry Pi to enable file sharing. Begin by installing Samba from the terminal by typing:

```
sudo apt-get install samba samba-common-bin
```

Then create a directory that you will share by typing:

```
mkdir /share
```

Samba's configuration information is stored in the file **/etc/samba/smb.conf**. Open this file with your favourite text editor and delete the contents. Now copy in the following text and save it:

```
[global]
netbios name = LaserCutter
server string = k40 pi
workgroup = WORKGROUP

[laser]
path = /share
comment = Laser cutter shared directory
browseable = yes
writeable = Yes
only guest = no
create mask = 0777
directory mask = 0777
Public = yes
Guest ok = yes
```

This will make a public shared directory on your machine that can be accessed by K40 Whisperer. Restart your Raspberry Pi and your drive should appear on the network. You can make K40 Whisperer go directly to this location by changing the value of `init_dir` in the functions `menu_File_Open_Design` and `menu_File_Open_EGV`. Changing the `init_dir = os.path.dirname(self.DESIGN_FILE)` to `init_dir = "/share"` will open the file selector in the shared directory when it is used.

If you are having problems connecting to the Samba share with Windows 10, you might need to enable SMB1.0/CIFS support by opening Control Panel > Turn Windows features on and off. In the list of options, find 'SMB1.0/CIFS File Sharing Support' and make sure it is enabled.

## QUICK TIP

You can skip steps one to six of the **README_linux.txt** file when installing K40 Whisperer, because there will only be one user in this installation, and controlling access to the laser is less important.

Wireframe

Join us as we lift the lid
on video games

ALL FORMATS
116 PAGES MONTHLY
Wireframe
LIFTING THE LID ON VIDEO GAMES
ENTER THE MATRIX
The inside story of a flawed, ambitious movie tie-in
BUILD A CITY
...using the engine behind Heaven's Vault
Issue 45 £6
wfmag.cc
Baldur's Gate III
LARIAN STUDIOS REVIVE A STONE-COLD CLASSIC
LAUNCH TITLES
EXPLORING THE BEST AND WORST GAMES FROM EACH GENERATION
TEAM17 AND THE 2
ROAD TO WORMS
TAKING SIDES
How to design better warring game factions

Visit **wfmag.cc** to learn more

HackSpace
TECHNOLOGY IN YOUR HANDS

# FIELD TEST

HACK | MAKE | BUILD | CREATE

Hacker gear poked, prodded, taken apart, and investigated

PG 108

DIRECT FROM SHENZHEN:
## HOT AIR STATION

Blast your circuits with scorching heat and melt metal

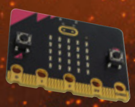
PG 110

## BANGLE.JS

A microcontroller you wear on your wrist

PG 112

## MICRO:BIT V2



The educational development board grows up

PG 102

## BEST OF BREED

Maker gaming systems

PLAYER 1
PLAYER 2
BUTTSNAP

ONLY THE BEST

# Gaming on the go!

Gaming systems that fit in your pocket

By **Marc de Vinck** @devinck

**It's that time of year when the console wars heat up.** PlayStation vs Xbox vs Stadia and more. All the big players have new systems and games hitting the shelves. The challenge is finding one! Gaming is a huge community, and we're not even including the incredibly popular PC crowd of gamers. But what happens when you are on the go? Or worse, when someone else is playing your system, and you have a need for some good old-fashioned competition? Well, that's where these portable game systems come into play.

> **Some are much smaller, and maybe a few push the limits of 'pocketable',** but they are all packed with fun

There are tons of DIY game kits out there, from the ever-popular Raspberry Pi and Retro Arcade scene, to dozens of different handhelds and desktop systems. In this Best of Breed, we'll be looking at the truly portable and affordable systems, both in kit form and fully assembled. The general specification is that they must fit in your pocket or measure around the same size as a deck of cards. Some are much smaller, and maybe a few push the limits of 'pocketable', but they are all packed with fun.

ARCADE
GAME ROOM

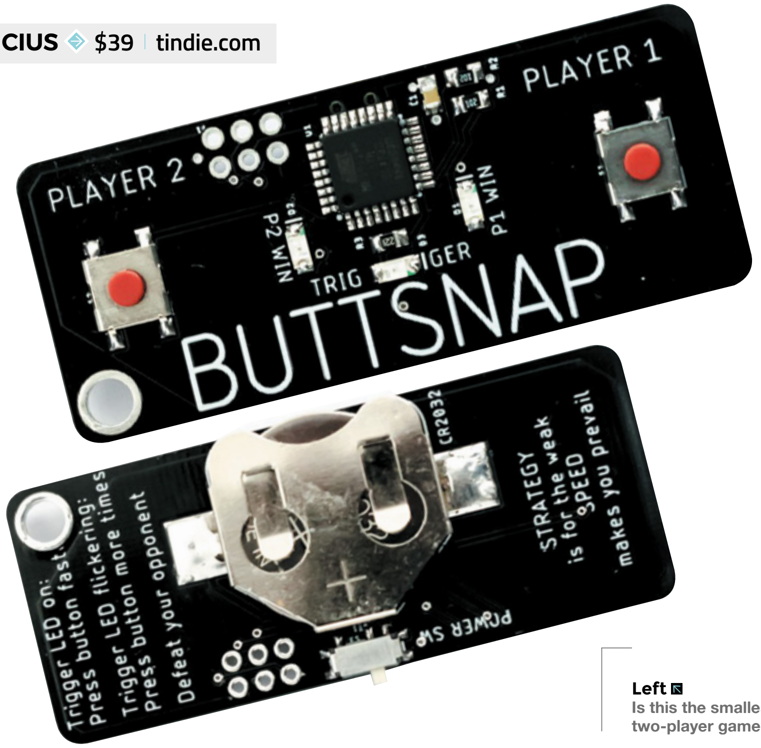
# ButtSnap vs LED Station Portable

**STUDIOBELOW** ◆ $15 | tindie.com

**BOBRICIUS** ◆ $39 | tindie.com

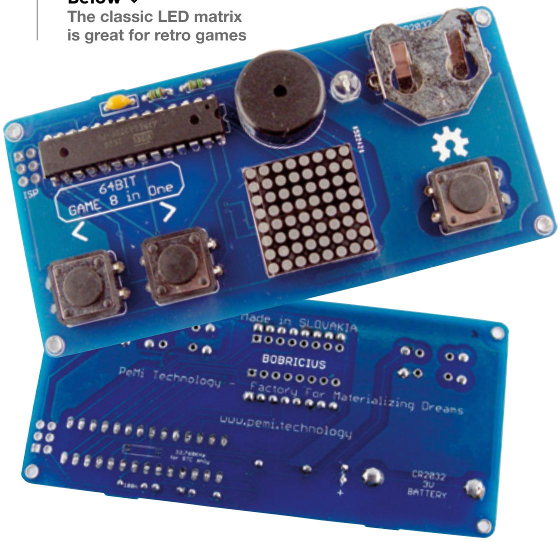
**Small? Check!** Fun? Check! Funny name? Double-check!! What is there to say about the ButtSnap, other than it's exactly the product we were looking for when it comes to portable competitive fun. It's truly simple, and most people won't even need to be told how to play. Simply wait for the LED to blink and beat your opponent's reaction time. But there is more…

What we really like about the ButtSnap is the unexpected features of the game. Every so often, a bonus speed round is started. The LED flashes rapidly for a fixed period of time, and the person who clicks their button the most in that time wins the round. A very clever way to add another dimension to an otherwise simple game. We have only one request for a future version: make it a kit! We'd love the opportunity to solder this together. But then again, many people will appreciate the fact that this comes ready for some button-mashing competition.


PLAYER 2
PLAYER 1
P2 WIN
P1 WIN
TRIGGER
BUTTSNAP
CR2032
STRATEGY is for the weak SPEED makes you prevail
Trigger LED on: Press button fast
Trigger LED flickering: Press button more times
Defeat your opponent
POWER SW

**Left**
Is this the smallest two-player game?

**Below** ◆
The classic LED matrix is great for retro games


64BIT GAME 8 in One
Made in SLOVAKIA
BOBRICIUS
PeMi Technology - Factory for Materializing Dreams
www.pemi.technology
CR2032 3V BATTERY

**The classic 8×8 LED matrix, loved by so many electronics enthusiasts, is the main feature of this minimal handheld game system by bobricius.** But don't be fooled – you can do a lot with just 64 LEDs and an ATmega328 IC. The LED Station can play eight different games, including modified versions of Tetris, Pong, Breakout, Snake, and more.

And since it's basically an Arduino at its core, you can fairly easily hack it and upload your own code. Head over to the website for a link to the source code, libraries, design files, and more. There are also a few good videos that demonstrate all the games included. We're always amazed at the games you can create with only three buttons! →

## VERDICT

ButtSnap

Two-button fun!

**10**/10

LED Station Portable

A bit pricey, but it comes assembled.

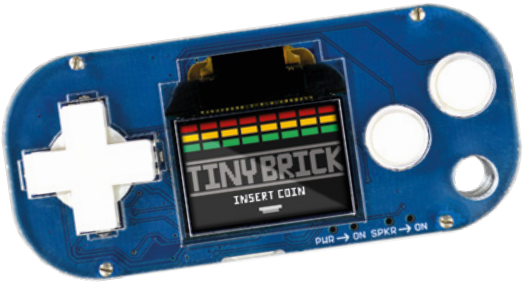
**7**/10

# Pocket Arcade

**TINYCIRCUITS** ◆ $59.95 | tinycircuits.com

**We've been a fan of the TinyCircuits ecosystem for many years.** This author remembers building a wrist-watch with a few simple clicks of their components while simultaneously heading out the door on a business trip. Everyone loved the 3D-printed case he designed, and they were amazed at the size and quality of the screen. Ever since then, we've been hooked on TinyCircuits.

The Pocket Arcade kit leverages the same amazing OLED colour screen and click-together technology as that watch, and packages it up with a 32-bit ARM processor, laser-cut case, power supply, and speaker. You can create your own games, download them from the TinyCircuits website, or even encode videos to watch on the go. This little kit can play hours of movies from an SD card, with audio, in surprisingly good resolution – check out their website for more information on the Pocket Arcade, and all the other components available for your next micro-build.



**Left** ◆
A lot of power in a small space

**VERDICT**

Pocket Arcade

Very well made and super-small.

**8** /10

* * *

# SparkFun Simon Says

**SPARKFUN** ◆ $26.95 | sparkfun.com

**A classic game of Simon Says designed by SparkFun as a great learn-to-solder kit.** This author has put dozens of these kits together at conferences, and it's one of our go-to kits for teaching anyone how to solder. SparkFun has great documentation to support the Simon Says kit too, including a brief introduction to soldering. It's another reason why we'd like to recommend this particular kit to people just starting out with electronics and soldering.

And, just like some of the other kits in this roundup, this kit features an ATmega328 at its core (the chip found in so many Arduino boards). It makes it surprisingly easy to hack, and learn more about, since the open-source ecosystem is so robust around this popular microcontroller. Speaking of hacking, check out SparkFun's instructions on the product page where they walk you through the process of reprogramming the ATmega and getting access to the piezo speaker and LEDs. It's a really fun introduction to programming and embedded electronics.

Simon Says
RAYOVAC
RAYOVAC
Power
Off
Off
Sound

**Left** ◆
Build a game while learning to solder

**VERDICT**

SparkFun Simon Says

A classic kit of a classic game.

**9** /10

# Adafruit PyGamer Starter Kit

**ADAFRUIT** ◆ $59.95 | adafruit.com

**Everything Adafruit designs and manufactures is well thought out and beautiful.** And the PyGamer Starter Kit is no exception. This open-source kit is capable of running your choice of CircuitPython, MakeCode Arcade, or Arduino code, making it incredibly easy and extensible. There is something for everyone, from new users to seasoned programmers.

> "There is something for everyone, from new users to **seasoned programmers**"

This is the kit version of the PyGamer, which includes the PyGamer PCB, case, batteries, and more for an affordable price. You can also pick up just the PyGamer PCB and other needed components separately, but the kit is a great choice, especially as a gift.

The PyGamer is powered by an ATSAMD51, which features 512kB of flash and 192kB of RAM. Adafruit then added an additional 8MB of QSPI flash for game file storage. It also features a 160×128 colour TFT display, a dual potentiometer analogue stick, four square-top buttons, and five NeoPixel LEDs. The back of the board has a socket that allows you to expand the system by plugging in any of their FeatherWing boards. There are a lot of other connectors and components packed on the PCB. Your best bet is to head on over to the website to get all the details. →

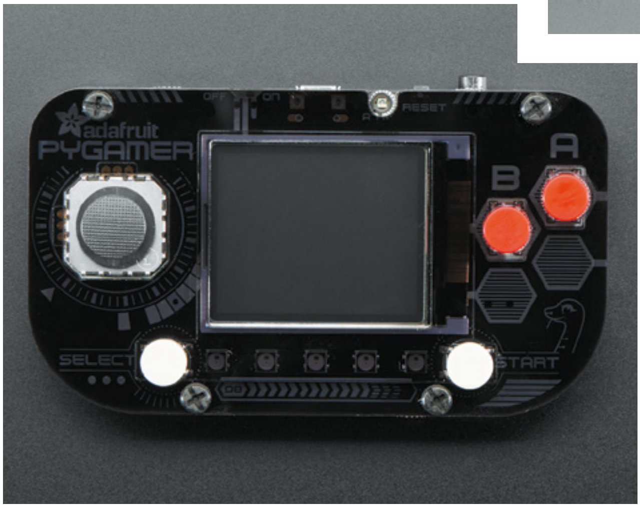
**Left** ↙
Code your games in Python

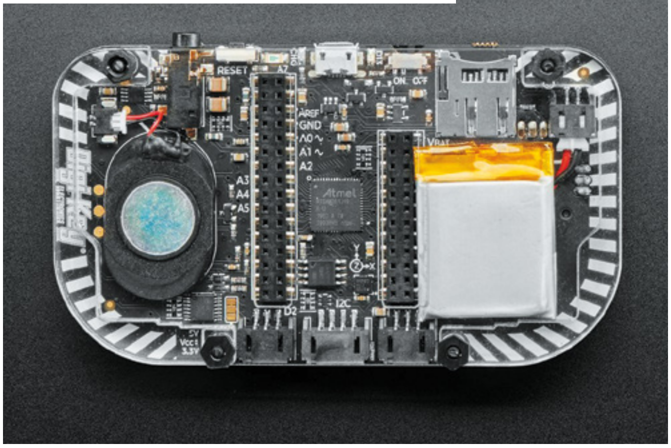
## VERDICT

Adafruit PyGamer Starter Kit

**Well made, and well played.**

**10**/10



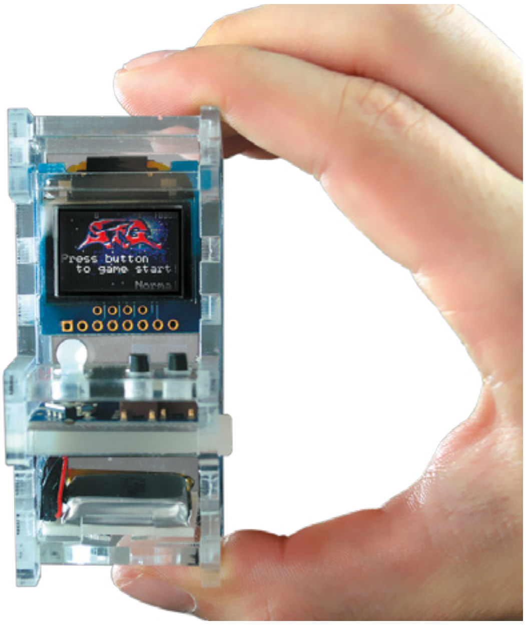


# Tiny Arcade DIY Kit

**TINYCIRCUITS** ◆ $59.95 | tinycircuits.com

**Another home run of an arcade kit from TinyCircuits!** This author has owned this kit for many years, and it still gets regular use by anyone visiting the studio. The Tiny Arcade DIY Kit comes with everything you need to make a miniature arcade game. At its core is a 32-bit ARM processor, a full-colour OLED screen, built-in speaker, and LiPo battery. All of this is housed in a laser-cut case with a four-way joystick and two buttons.

This is one of those kits that you need to see to believe. The responsiveness of the joystick, coupled with the clarity of the OLED screen is amazing. No, it's not going to replace your next-gen console gaming system, but you'd be surprised that it often gets more attention. It's a fun little kit that anyone can assemble, no soldering required.



**Left** ◆
Create your own mini arcade

**VERDICT**

Tiny Arcade DIY Kit

One of our favourite gaming kits.

**10**/10

# Espresso Kit

**AMPERSAND** ◆ $47 | **tindie.com**

**The Espresso is a DIY gaming system that uses the ever-popular form factor of the full-size Game Boy Pocket buttons, coupled with a 2.2-inch screen, an ESP32, and a clever and beautifully designed ten-PCB enclosure.** Yes, ten PCBs! Thanks to the ESP32, you can play many of your favourite retro arcade system games, including the NES, Game Boy, Sega Master System, ColecoVision, and more.

What really caught our attention is the ten PCBs that make up the case. By stacking them together, you get a very robust and beautiful enclosure. You also get a great way of adding in all the other components, since you can have traces and connectors wherever you need them!

B
A
MENU
VOLUME
SELECT
START

The kit includes most of what you need, but you will have to supply the battery, a microSD card, and Game Boy Pocket buttons. We're not sure why the buttons aren't included, as they can be sourced fairly easily for only a few dollars, but being able to reuse buttons from a beloved Game Boy system or personalise the colour with a new set is nice too. □

**Left** ◆
Just add buttons!

**VERDICT**

Espresso Kit

Well worth the semi-complicated build.

**8**/10

# GET STARTED WITH ARDUINO

Robots, musical instruments, smart displays and more

£10 with FREE worldwide shipping

**Inside:**

- Build a four-legged walking robot
- Create a Tetris-inspired clock
- Grow veg with hydroponics
- And much more!

AVAILABLE NOW hsmag.cc/store

plus all good newsagents and:

WHSmith BARNES&NOBLE

Available on the App Store

GET IT ON Google Play

FROM THE MAKERS OF HackSpace MAGAZINE

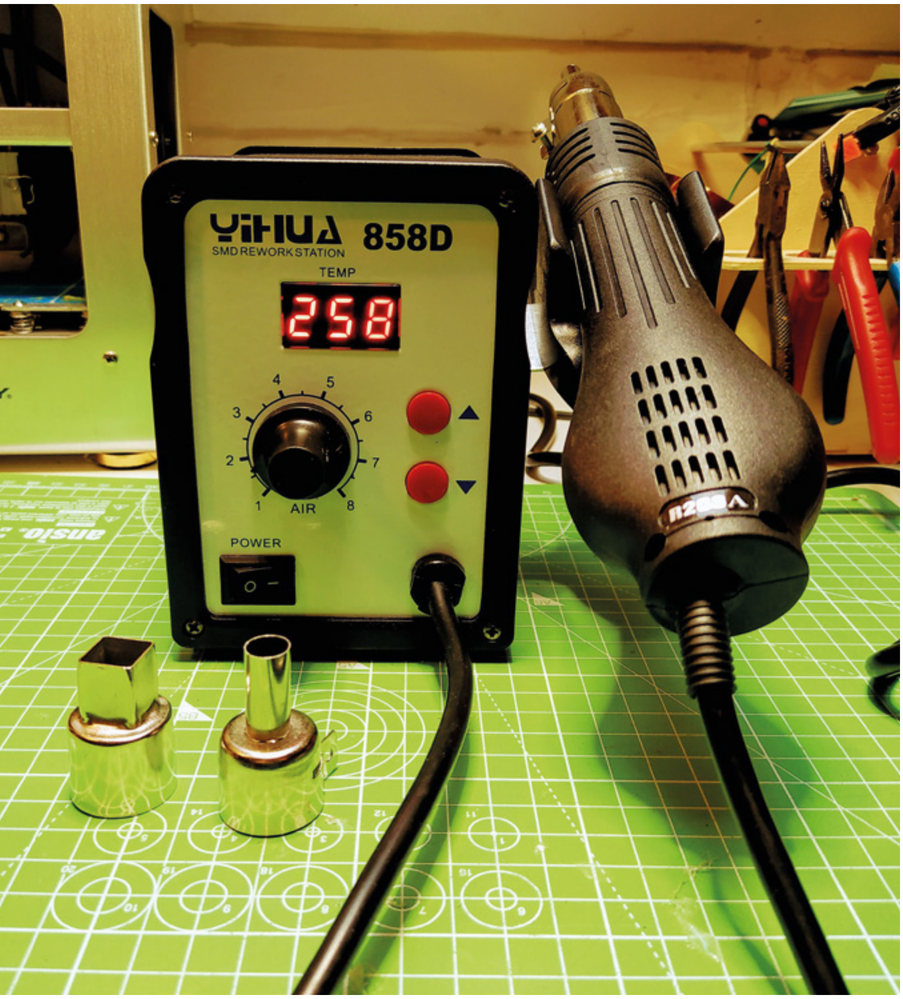
GET STARTED WITH
ARDUINO
Robots, musical instruments, smart displays and more
Create AMAZING projects with this programmable controller
FROM THE MAKERS OF HackSpace MAGAZINE

# Hot air reflow station

A fancy hair-dryer or useful soldering kit?

**By Jo Hinchliffe** @concreted0g

**Below** The tips shape the hot air blowing out

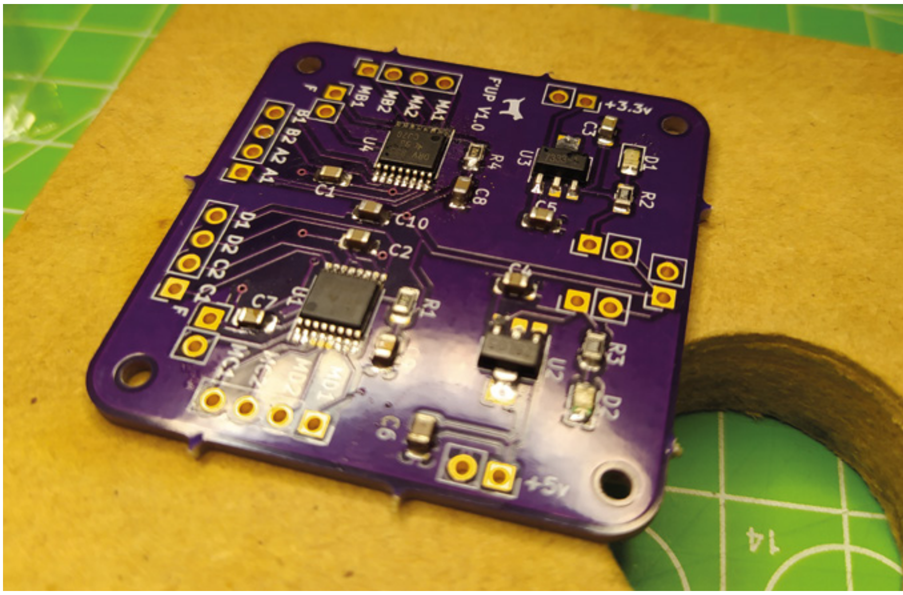


**Hot air reflow work is a valuable technique to be able to access at home.** It is excellent in terms of being able to populate PCBs with smaller SMD components, but also incredibly useful in removing items from PCBs for repair, or for salvaging components off boards. There are lots of cheap hot air reflow tools available and many are bundled together with both a hot air tool and a conventional soldering iron. Lots of people, ourselves included, will have one or more soldering irons, so we were interested to explore the cheap 858D station which features hot air only.

The 858D is cheap and cheerful, and online searches reveal so many differently badged items. With so many clones, it's unclear which versions are the originals. There are also lots of examples of people modifying the 858D and, in certain cases, rectifying some safety concerns, more of which later!

At £32, our 858D was branded 'Yihua', and arrived from China well-packed in a cardboard box with decent foam packing. Inside the box, the 858D was found with the correct UK mains plug attached, and shipped with three different pattern end pieces for the hot air gun. We'd read online about many 858Ds being shipped where the hot air gun part was not connected to the ground plane, and that many people ended up creating a wire link between the gun and the ground connection on the chassis. Obviously it's a concern if the gun isn't grounded as, in the event of a fault, there is a chance that the metal end of the gun could become energised.

A simple check is to grab a multimeter capable of performing a continuity check and, before plugging the machine in, put one probe on the end of the gun and

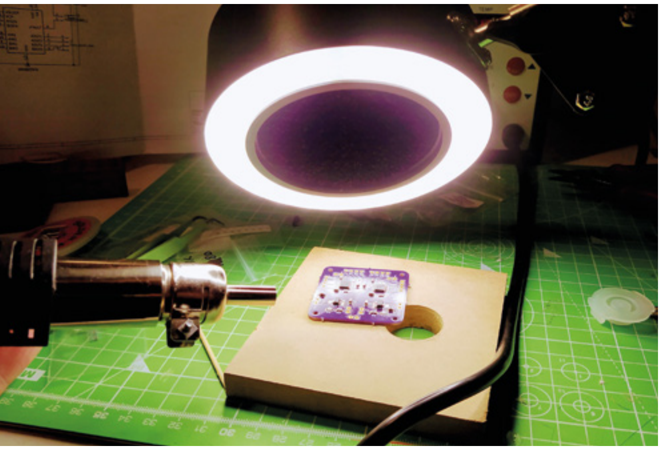


**Above** ◆
We were able to solder these surface mount components using this station

touch the other probe to the ground pin on the mains plug. Doing this check showed that our machine was indeed grounded through to the tip of the gun. We'd also read online a worrying story that the fuse holders on some 858Ds are in-line on the neutral line rather than the live mains line between the mains input and the controller PCB. We had also read and seen a few photos of other wiring and connector issues, and so we decided to inspect the internals before powering on the unit. We opened up the box and were pleased to see that the fuse was wired correctly on our unit, and we were actually impressed with the general state of the wiring and connectivity. We'd seen a large variety of internal pictures online, as many of these 858D units have differing ICs and PCBs and numerous different modifications and firmwares have been developed by the community. We noted that our main IC was directly soldered to the PCB rather than in a socket, which may create an amazing chicken and egg situation if we ever wanted to replace it, as we'd need hot air reflow to remove it!

Reassured that everything was as it should be, we powered up the unit. The controls are simple, with a knob to turn that increases and decreases the air flow through the gun, and buttons that increase or decrease the temperature. There is a small holder for the air gun mounted on the side of the unit, and it has a magnet and reed switch arrangement to detect when the gun is in the holder. Dialling in a temperature and increasing the air flow, the gun only begins to heat when removed from the holder. Similarly, once up to temperature, if the gun is replaced in the holder, the fan continues to run until the temperature is reduced to 100 degrees and then the fan cuts out. This speeds cooling, but it's still important to make sure that the hot air gun is not close to any materials that it could burn whilst cooling. Lifting the gun, the temperature increases and the fan starts again. It's a matter of seconds for the unit to return to its operational temperature and isn't frustrating in terms of use.

**Below** ◆
The 858D is easy to hold and use, and sits comfortably in the hand.



Using a clean PCB, we applied some tiny amounts of solder paste and populated a board. Making sure our fume extractor was running and our workspace was also adequately ventilated, we fitted a small circular nozzle to the hot air gun. Moving the hot air gun over the part laid on the solder paste, we quickly reflowed the part in a few seconds. If you haven't used hot air reflow before, it's incredibly satisfying to see nice shiny joints materialise in front of your eyes magically! We opted to use the smallest circular nozzle, and we imagine this will sit on the machine most of the time, unless we are trying to remove a large component. In use, the hot air gun is comfy in the hand and not unwieldy at all. We tended to have the air flow set at the lower end of the scale, as increasing it makes it very easy to blow small SMD components like resistors and caps off the paste. Set between 375 and 400°C, our air gun struck a good balance of heating to the reflow temperature quickly, without cooking the board, and we got good results with our solder paste.

For the price point, and bearing in mind our unit didn't have any of the safety concerns others have reported, it's hard to find any complaints with our 858D, and its small footprint sits nicely on the desk, waiting to be called into service. □

# Bangle.js

A smartwatch that you can control

**ESPRUINO** ◆ £69.96 | **banglejs.com**

By **Ben Everard** @ben_everard

**We're fans of hackable smartwatches here at HackSpace towers.** They offer a tantalising chance to build a future of IoT that works for us, not some global mega corp that wants to harvest our data or lock us into an ecosystem of ever increasingly expensive devices. We've previously looked at the LILYGO T-Watch-2020, and now we've had a chance to play with a Bangle.js.

N
W
E
S
100
11:55
04
Dec 3 2020

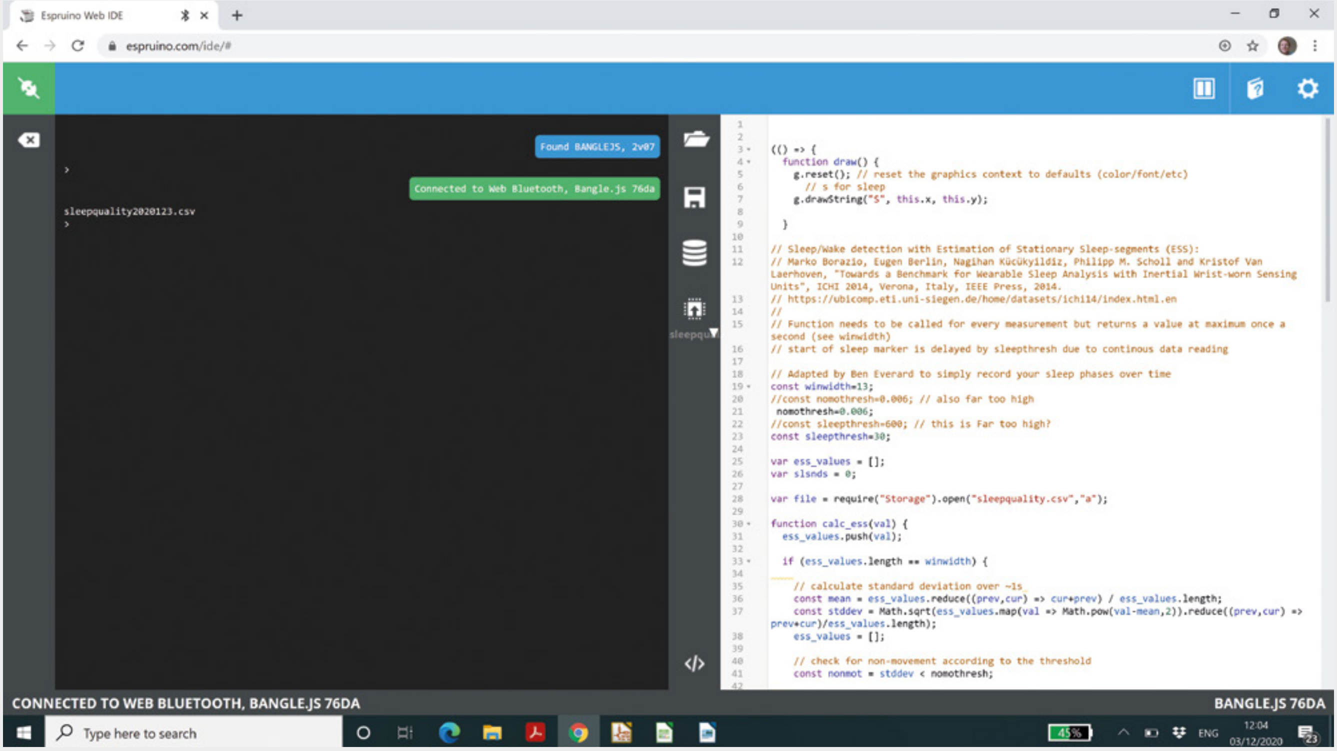
**Below** ◆ The watch is large, but does sit comfortably on the wrist (unless you're wearing a shirt)

As you may have guessed from the name, the Bangle.js is programmed in JavaScript. It also features something of a kitchen sink approach to hardware and is unbelievably packed with features for a wrist-borne device. There's a GPS receiver, a heart rate monitor, a three-axis accelerometer, a three-axis magnetometer, a vibration sensor, a 64MHz ARM Cortex M4 with 64Kb of RAM, 4MB of external flash, and a 350 mAh battery. On top of all this, it's waterproof for ten metres. It will come as no surprise that with all this on board, it's a bit of a whopper. It comes in a 5×5×1.7 cm case. This is significantly larger than any watch this author has worn before, and it's a visible statement piece on your wrist. If you wear shirts, you may find that it interferes with your cuffs. That said, we didn't find it uncomfortable to wear and it felt quite unobtrusive despite its size.

> " It features something of a kitchen sink approach to hardware and is unbelievably packed with features "

The Espruino firmware is open-source, so you can code for it at almost any level you like, but for most users, programming the Bangle.js will mean writing apps or widgets. These are very similar, other than the fact that widgets run in the background and apps in the foreground (apps can include widgets, if you need both functions).

You can develop these on the online Espruino IDE at **espruino.com/ide**. From there, you can upload your code over Bluetooth (there's no data-in on the watch, and the USB charge cable is power only). This



```
>

sleepquality2020123.csv
>
```

Found BANGLEJS, 2v07

Connected to Web Bluetooth, Bangle.js 76da

```


(() => {
  function draw() {
    g.reset(); // reset the graphics context to defaults (color/font/etc)
      // s for sleep
    g.drawString("S", this.x, this.y);

  }

// Sleep/Wake detection with Estimation of Stationary Sleep-segments (ESS):
// Marko Borazio, Eugen Berlin, Nagihan Kücükyildiz, Philipp M. Scholl and Kristof Van
Laerhoven, "Towards a Benchmark for Wearable Sleep Analysis with Inertial Wrist-worn Sensing
Units", ICHI 2014, Verona, Italy, IEEE Press, 2014.
// https://ubicomp.eti.uni-siegen.de/home/datasets/ichi14/index.html.en
//
// Function needs to be called for every measurement but returns a value at maximum once a
second (see winwidth)
// start of sleep marker is delayed by sleepthresh due to continous data reading

// Adapted by Ben Everard to simply record your sleep phases over time
const winwidth=13;
//const nomothresh=0.006; // also far too high
 nomothresh=0.006;
//const sleepthresh=600; // this is Far too high?
const sleepthresh=30;

var ess_values = [];
var slsnds = 0;

var file = require("Storage").open("sleepquality.csv","a");

function calc_ess(val) {
  ess_values.push(val);

  if (ess_values.length == winwidth) {

    // calculate standard deviation over ~1s
    const mean = ess_values.reduce((prev,cur) => cur+prev) / ess_values.length;
    const stddev = Math.sqrt(ess_values.map(val => Math.pow(val-mean,2)).reduce((prev,cur) =>
prev+cur)/ess_values.length);
    ess_values = [];

    // check for non-movement according to the threshold
    const nonmot = stddev < nomothresh;
```

CONNECTED TO WEB BLUETOOTH, BANGLE.JS 76DA

BANGLE.JS 76DA

**Above** ◆ The Espruino IDE lets you upload code, interact with the serial console, and pull data off the watch

reviewer has done some JavaScript development over the years, but only a little, and he found it straightforward to get started. Perhaps the main thing to get your head around is the event-driven nature of the language. Lots of things are driven by callbacks to functions passed as parameters. This sounds more complex than it is. Really, it just means that you create functions that you want to run when particular events happen, and then tell the watch which event you want to trigger your function. An advantage of this is that it means that the firmware can take care of power management. In your code, you just link in what you want to happen when – and when nothing's running, it can make sure that power isn't wasted.

The IDE provides the ability to grab data files off the watch, which is a great way of working with anything that tracks data. However, it's worth bearing in mind that Bluetooth is the only way of getting data on or off the watch. You can download the files manually like this, or you could write a companion app (or website) that gets the data you want. It's possible to connect to the device with another Bluetooth-compatible device using Puck.js to stream data to a web service if you want, but this isn't entirely straightforward.

There's an app store of existing applications at **banglejs.com/apps**. This is driven by GitHub as the data store, so if you want to see how an app works, just click on the GitHub icon and you can see the source code. What's more, this is a static page driven by GitHub Pages, so you can fork this, enable GitHub Pages, and create your own app store to which you can add your apps (see this reviewer's at **benevpi.github.io/BangleApps**). We've been working on a sleep quality tracker, which you can see in this store. Once you're happy with your code, if you want, you can submit a pull request back to the original repository and your app will be listed on the global app store.

GPS reception is OK. Indoors, it can struggle, but outdoors, we found it worked better. This is probably because antennae don't like being crammed in a small space with lots of other electronics, especially when they're trying to pick up weak signals from spacecraft thousands of miles away.

We also found the heart rate monitor prone to suspect readings. We found we could get accurate data by taking five readings, then selecting the middle one, but obviously this limits the speed at which you can get data.

These minor gripes aside, we found the Bangle.js worked excellently. We were able to play with other people's apps, poke around their internals, and code our own – exactly what you'd expect to be able to do with an open-source watch. For the price, this is an excellent development platform for personalised electronics, with more features than we know what to do with. We've already got some plans for this watch – keep an eye on the mag for future articles. However, it is a big watch and right on the limit of what we'd consider an acceptable size for a wrist-based device. ☐

## VERDICT

A fun and versatile wrist-based computer.

**9**/10

# micro:bit version 2

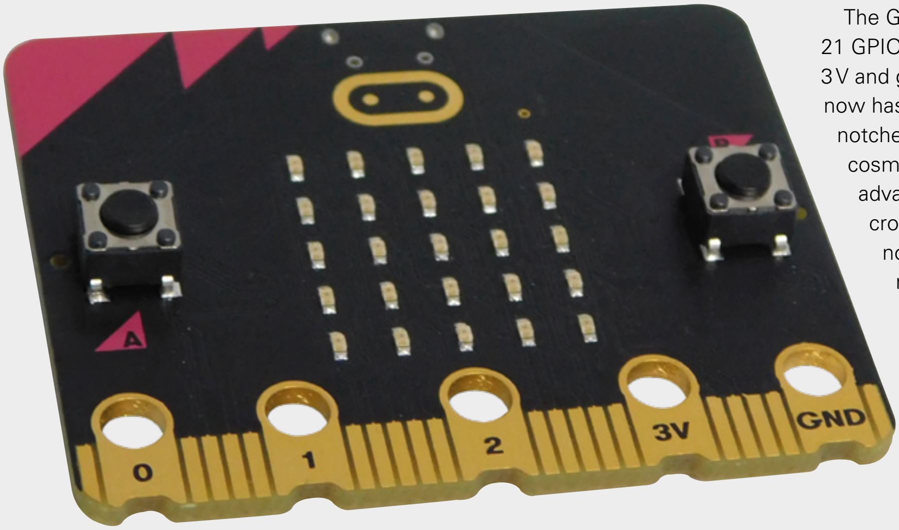
The educational microcontroller gets an upgrade

**MICRO:BIT EDUCATIONAL FOUNDATION** ◆ £13.50 | microbit.org

By **Ben Everard** @ben_everard

**Since 2016, the BBC micro:bit has been a fixture of the UK tech education scene.** The little board, with its iconic array of 25 red LEDs and two user buttons, can be found in schools and youth groups up and down the country. It's got chunky GPIO pins that are easy to connect to, yet is still small enough to fit in little hands and feel comfortable to use. The micro:bit has introduced a generation of children to microcontrollers and programming. Four years after the initial release, the Micro:bit Educational Foundation has announced a major upgrade to this little board. Let's take a look at this new micro:bit, and what it means for users.

**Below** ◆ The front of the new micro:bit will be instantly recognisable to anyone familiar with the form factor

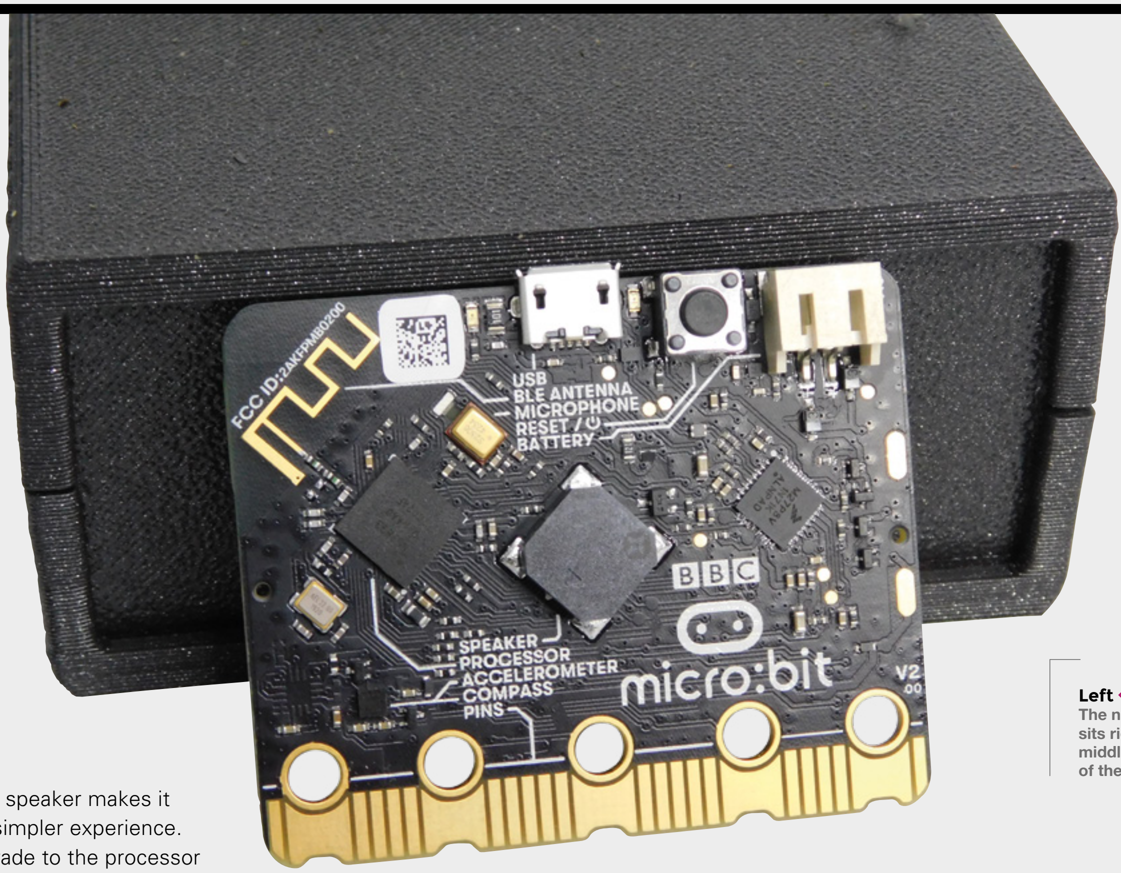


Before looking at the shiny new things, let's take a quick look at what's still the same. There's still a 5×5 grid of red LEDs, and two buttons. There's still an accelerometer, compass, and temperature sensor. The programming environments are also still the same, and code for your micro:bit v1 should work without alteration on a v2 board.

So, if all that's the same, what's new? There's a significant boost to processing power, as the new Nordic nRF52833 microcontroller boasts a 64MHz ARM Cortex M4F (up from a 16MHz ARM Cortex M0+ in the previous micro:bit). There's now 128kB of RAM and 512kB of storage. Bluetooth is upgraded from 4.0 to 5.0. For audio, there's a microphone and speaker, and the logo at the top of the board is now touch-sensitive.

The GPIO connector is almost the same. It still has 21 GPIOs, with three broken out as large pads, and 3V and ground connectors. However, the connector now has notches on each of the large pads. The notches in the GPIO connector may seem like a cosmetic change, but they do offer a significant advantage of making it easier to connect with crocodile clips or conductive thread. Clips can now go perpendicular to the board, and the notch will stop them from slipping sideways to bridge an additional GPIO.

The changes open up significant new ways you can work with micro:bit. Sound has long been a popular tool of teachers getting students interested in coding and, while the previous micro:bit could make sounds, you had to manually connect headphones using crocodile clips.



**Left**
The new speaker sits right in the middle of the back of the micro:bit

Adding a speaker makes it a much simpler experience. The upgrade to the processor also brings machine learning within range. TensorFlow Lite can run on microcontrollers such as this, and the Edge Impulse team have already demonstrated keyword recognition from voice running on the new board (see here for details: **hsmag.cc/EdgeImp**).

The one major downside to the micro:bit v2 is, for us, the same as the one major downside to the micro:bit v1 – connectivity. There are three large pins broken out and this doesn't really give you much scope for anything beyond adding a few buttons.

> For a hobbyist, there are undoubtedly some great **projects you can build with this**

There are an additional 18 GPIOs available on the edge connector, but you will need an adapter to make this work. Of these 18, ten are needed for internal use – the LED array, buttons, and I2C bus are all included in this count, so even with an edge connector, there are relatively few connections available to use unless you disable these functions. There's also no 5V (or VBUS), so if you want to use hardware that needs this level supply, you will need an external power supply.

However, if you want to expand the capabilities of your micro:bit, there's a good selection of add-ons available from a range of manufacturers. You can use your micro:bit to control a robot, water your plants, or play games – all by clipping it into off-the-shelf hardware. All these should continue to work with the new board.

For an educational microcontroller board, the advantage of the micro:bit really has nothing to do with the hardware – it's the set of resources that go with this. Professionally made and curated projects and lesson plans can be found at **microbit.org**. These, on their own, are more important than the processor speed or amount of storage. The fact that the new device is compatible with code for the old micro:bit means that all the effort that went into creating them the first time is still paying off.

For a hobbyist, there are undoubtedly some great projects you can build with this board. However, if you need to use more than three GPIOs, you'll end up with a bulky project, and even accounting for this, you're still quite limited with connections. If you can live with this, the micro:bit version 2 represents excellent value for money.

For just over £10, you get a powerful microcontroller with a bunch of sensors and a couple of output devices (the LED array and the speaker). We don't know of any other development board that offers so much for such a small price. ☐

## VERDICT

**Cheap, powerful, and feature-packed, but limited connectivity.**

**9**/10

# next month

issue #40

ON SALE
**18 FEBRUARY**

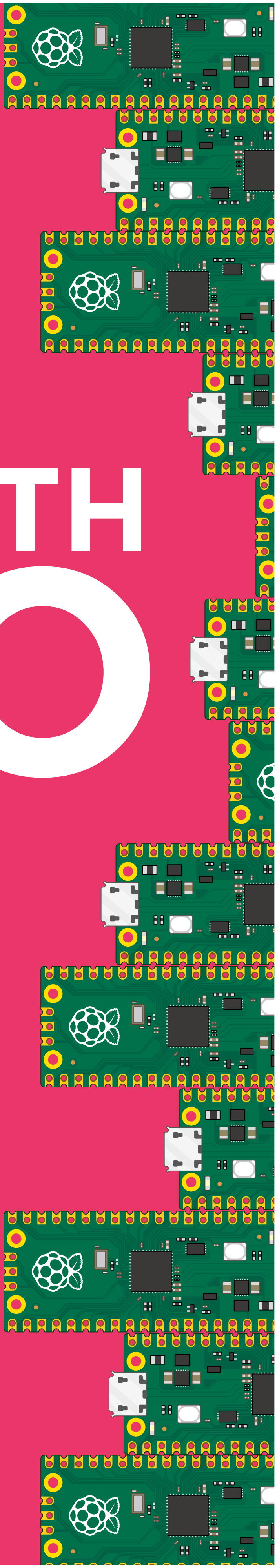
# BUILD WITH PICO

## ALSO

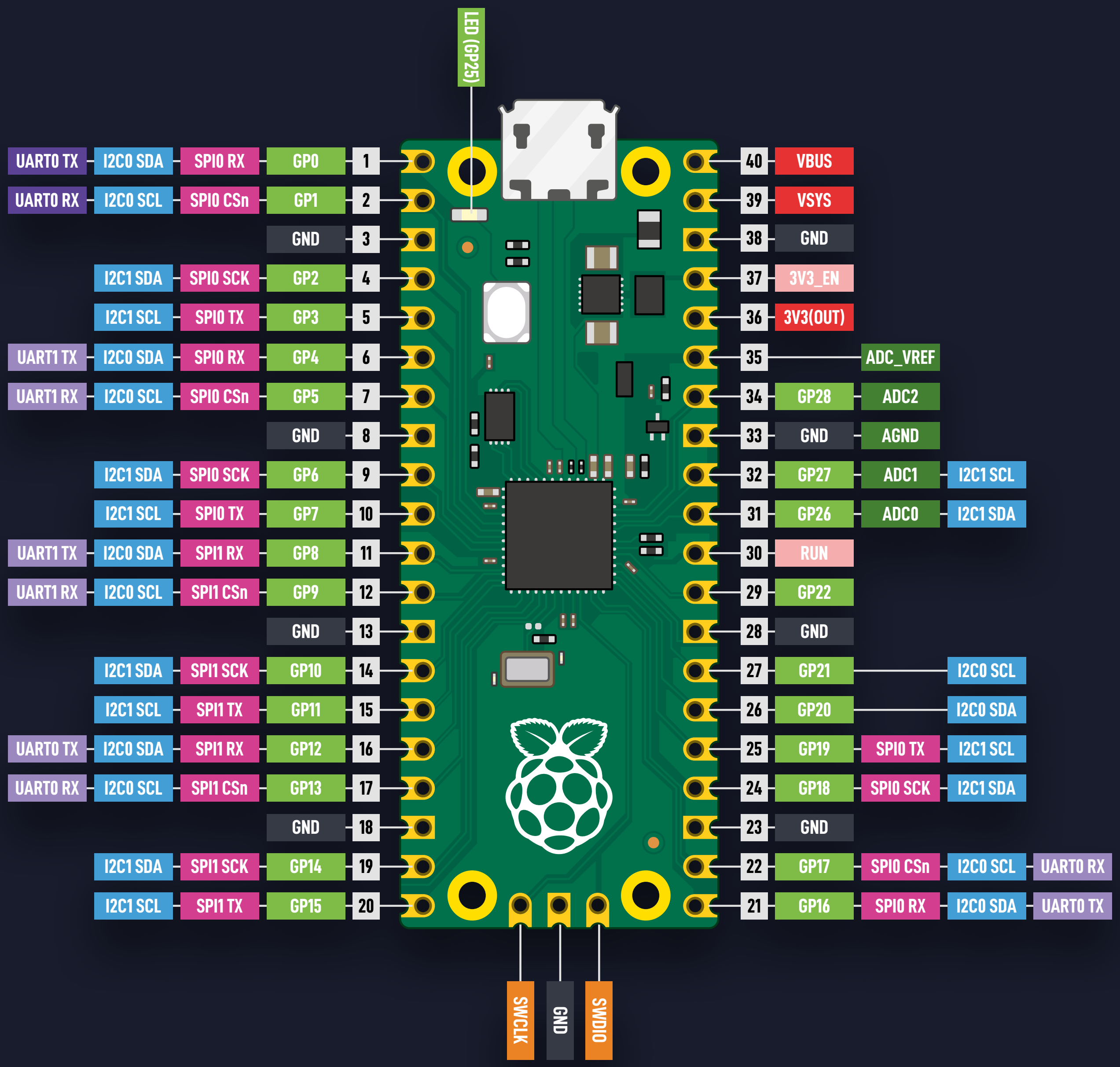
- → LASER CUTTING
- → MUSIC
- → 3D PRINTING
- → DIY SMARTWATCH
- → AND MUCH MORE

**DON'T MISS OUT**

hsmag.cc/subscribe



# Raspberry Pi Pico Pinout



| | | | | |
|---|---|---|---|---|
| UART0 TX | I2C0 SDA | SPI0 RX | GP0 | 1 |
| UART0 RX | I2C0 SCL | SPI0 CSn | GP1 | 2 |
| | | | GND | 3 |
| | I2C1 SDA | SPI0 SCK | GP2 | 4 |
| | I2C1 SCL | SPI0 TX | GP3 | 5 |
| UART1 TX | I2C0 SDA | SPI0 RX | GP4 | 6 |
| UART1 RX | I2C0 SCL | SPI0 CSn | GP5 | 7 |
| | | | GND | 8 |
| | I2C1 SDA | SPI0 SCK | GP6 | 9 |
| | I2C1 SCL | SPI0 TX | GP7 | 10 |
| UART1 TX | I2C0 SDA | SPI1 RX | GP8 | 11 |
| UART1 RX | I2C0 SCL | SPI1 CSn | GP9 | 12 |
| | | | GND | 13 |
| | I2C1 SDA | SPI1 SCK | GP10 | 14 |
| | I2C1 SCL | SPI1 TX | GP11 | 15 |
| UART0 TX | I2C0 SDA | SPI1 RX | GP12 | 16 |
| UART0 RX | I2C0 SCL | SPI1 CSn | GP13 | 17 |
| | | | GND | 18 |
| | I2C1 SDA | SPI1 SCK | GP14 | 19 |
| | I2C1 SCL | SPI1 TX | GP15 | 20 |

| | | | | |
|---|---|---|---|---|
| 40 | VBUS | | | |
| 39 | VSYS | | | |
| 38 | GND | | | |
| 37 | 3V3_EN | | | |
| 36 | 3V3(OUT) | | | |
| 35 | | ADC_VREF | | |
| 34 | GP28 | ADC2 | | |
| 33 | GND | AGND | | |
| 32 | GP27 | ADC1 | I2C1 SCL | |
| 31 | GP26 | ADC0 | I2C1 SDA | |
| 30 | RUN | | | |
| 29 | GP22 | | | |
| 28 | GND | | | |
| 27 | GP21 | | I2C0 SCL | |
| 26 | GP20 | | I2C0 SDA | |
| 25 | GP19 | SPI0 TX | I2C1 SCL | |
| 24 | GP18 | SPI0 SCK | I2C1 SDA | |
| 23 | GND | | | |
| 22 | GP17 | SPI0 CSn | I2C0 SCL | UART0 RX |
| 21 | GP16 | SPI0 RX | I2C0 SDA | UART0 TX |

POWER
GROUND
UART
UART (default)
GPIO, PIO & PWM
ADC
SPI
I2C
SYSTEM CONTROL
DEBUGGING

PicoSystem

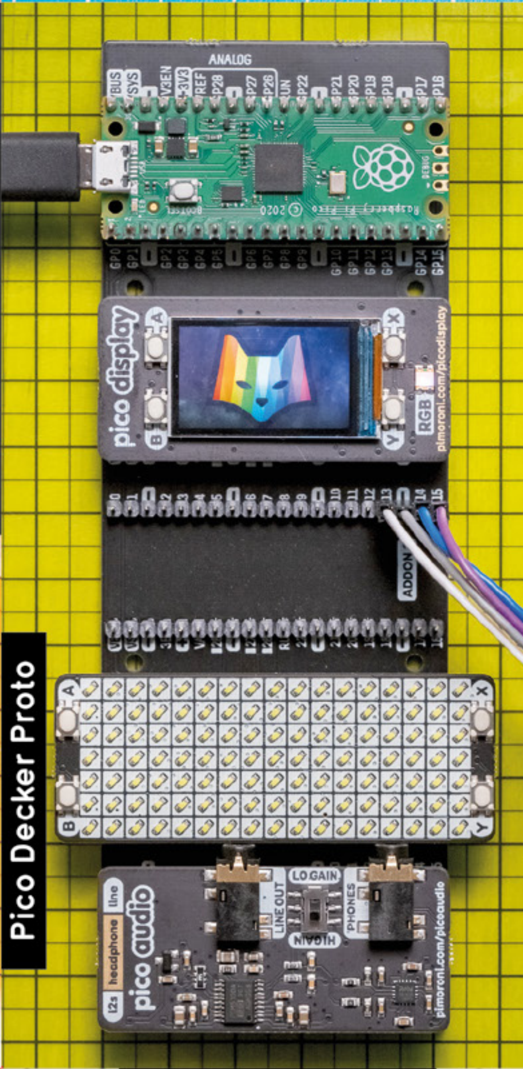
Pico Scroll Pack

Pico Audio Pack

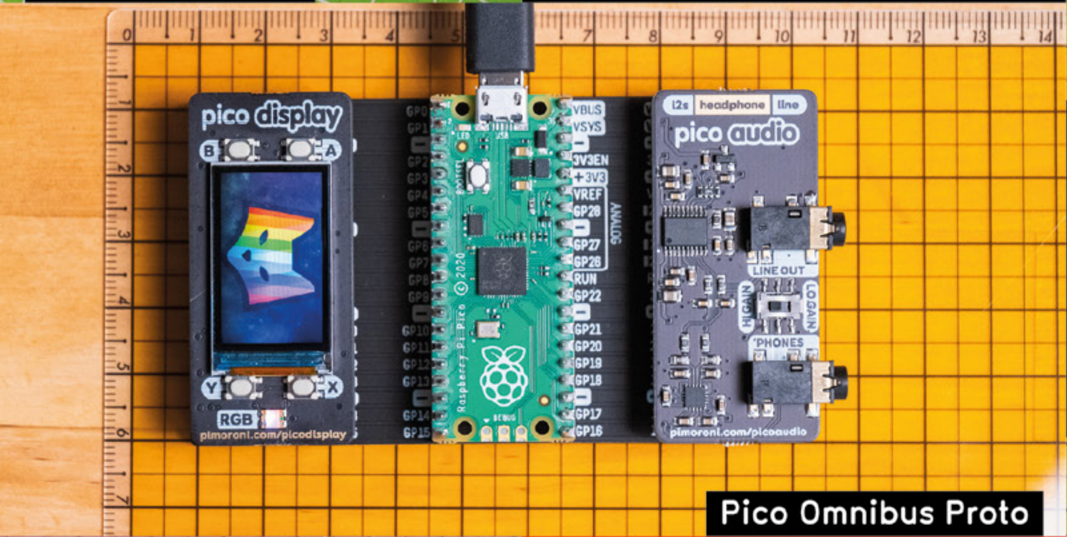
## full pico ecosystem

https://pimoroni.com/pico

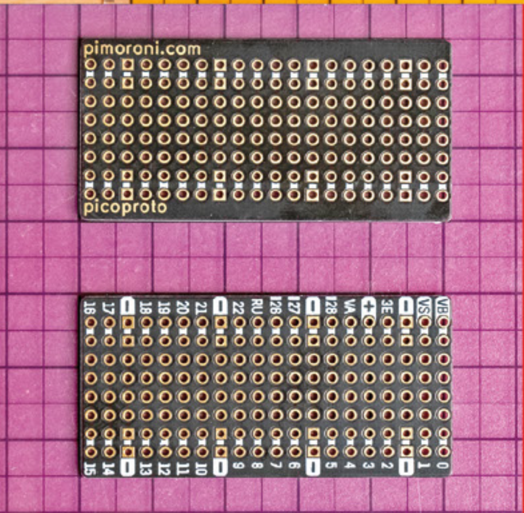
PIMORONI



Pico Decker Proto



Pico Omnibus Proto



We have a range of boards to enable your microcontroller tinkering, alongside our usual range of breakouts, breakboards and addons.

| PICO SHIM | PICO PACK | PICO BASE | PICO |
|---|---|---|---|
| minimal helper boards | fit neatly on the back of a Pico | multi-function. put Pico on top | host board in the Pico format |

Pico Display Pack